Chapter V

HARMONIC ANALYSIS OF SWITCHING FUNCTIONS

ROBERT J. LECHNER

I. SUMMARY

II. SURVEY OF ABSTRACT HARMONIC ANALYSIS
   A. The Fourier Transform of a Function on $\mathbb{Z}^n$ 125
   B. Survey of Classical Properties 139
   C. Fundamental Theorem on Invariance 148

III. COMBINATORIAL APPLICATIONS
   A. A Test for Implicants of $f$ or $\bar{f}$ 156
   B. Algorithm to Extract Prime Implicants 163
   C. Test for Disjunctive Decomposition 171
   D. Criteria for Factoring Variables 172

IV. ANALYSIS OF THE PROTOTYPE EQUIVALENCE RELATION
   A. Definition, Interpretation, and Context 174
   B. Enumeration of Prototype Equivalence Classes 183
   C. Explicit Definition of 5-Variable Prototypes 191

V. SYNTHESIS OF ENCODED INPUT LOGIC
   A. Review of LSI Logic Design Approaches 197
   B. Synthesis Techniques for Encoded Input Logic 204
   C. A 6-Variable Example 210
   D. Fundamental Problems and Extensions 218

REFERENCES 225

ABSTRACT. This chapter is a self-contained development of abstract harmonic analysis applied to a single-output combinational logic functions; linear algebra and elementary group theory are the only mathematical prerequisites. New analysis and synthesis techniques are developed, and the groundwork is laid for future extensions to multiple output combinational logic, sequential machines, and real- or complex-valued functions of binary arguments. Harmonic analysis adds novel conceptual insights and unifying principles, improved computational techniques, and new measures of complexity to the traditional approach to switching theory.

The first section is a summary of the chapter. Section II surveys classical Fourier transform properties and introduces the canonical expansion of a switching function as an
n-dimensional abstract Fourier transform over the finite two-element field. The two most important transform properties are the convolution theorem, which leads to tests for prime implicants and disjunctive decompositions, and spectrum invariance which is basic to further theoretical developments and to a new synthesis technique called encoded input logic.

Section III develops a new algorithm which concurrently extracts prime implicants and detects disjunctive decompositions of a switching function. Implicants of both the function and its complement are detected simultaneously, and "core" implicants can be identified. The algorithm which is not sensitive to functional complexity has been programmed for a commercial time-sharing system.

Section IV introduces the restricted affine group (RAG) whose elements, called prototype transformations, encode the arguments and outputs of combinational logic functions. This group partitions the space \( \mathcal{F} \) of all two-valued functions on \( \mathbb{Z}^n \) into 3, 8, and 48 equivalence classes respectively for \( n = 3, 4, \) and 5. Unique representatives are identified for each class when \( n = 3 \) and 4 and for 46 of the 48 classes of 5-argument functions.

Section V applies the tools of abstract harmonic analysis to the synthesis problem for large truth tables (many-input combinational logic). A general multilevel synthesis approach, called encoded input logic, is introduced which is compatible with large-scale-integrated circuit technology. Both the conventional macrocellular and the newer microcellular array approach are included as special cases. Prototype encoding transformations are used to reduce the complexity of an imbedded normal form realization. Practical synthesis algorithms are based on Fourier analysis. A realistic 6-argument example is treated in detail. Section V concludes with a list of fundamental problems whose solutions would extend the research presented herein.

I. SUMMARY

Harmonic analysis is a new method for both theoretical analysis and practical synthesis of digital logic circuits. This chapter is a simplified exposition which treats fully the single-output combinational logic case. It thereby lays the groundwork for future extensions, first to multiple-output combinational logic and then to sequential machines or finite state automata.

The organization of this chapter was dictated by its two-fold purpose. One goal is to assist the engineering-oriented graduate student or logic designer in applying harmonic analysis techniques to logic design for the technological environment of large-scale-integrated semiconductor electronics. Another goal is to develop the theoretical properties of abstract Fourier transforms for the mathematically oriented graduate student or logic theoretician as a point of departure for further research.

These two goals are somewhat inconsistent. The first goal demands attention to those computational properties of Fourier transforms which are directly applicable to combinational logic design. These should be introduced with a minimum of theoretical prerequisites. The second goal requires an abstract development which introduces all the known theoretical properties of Fourier transforms including those which appear to have no direct application, to stimulate new directions of research.
This chapter treats both theoretical and practical aspects of abstract harmonic analysis, with particular emphasis on discrete-valued functions of binary arguments. Of course, extensions to continuous real or complex-valued functions are possible and have considerable practical interest. Typical applications include reliability analysis for discrete systems and modulation theory for digital communications.

Basic material on harmonic analysis is introduced first, in Section II. An introductory section (II,A) is followed by a survey of classical Fourier transform properties (II,B). The representation of a switching function as an n-dimensional abstract Fourier transform over the finite two-element field is unique, therefore canonical, and has many valuable properties. These properties have inspired new algorithms for some classical problems of combinational logic synthesis to be described in Section III.

The most important theoretical contributions of harmonic analysis to switching theory depend on the intimate connection between Fourier transforms and linear or affine operators (encoding transformations) on the domain and range of switching functions. The fundamental theorem on invariance of Fourier transforms under linear and affine operators is introduced in Section II,C. Although this section is not a prerequisite to the engineering application of Section III, it is basic to the analysis of equivalence classes under transformation groups in Section IV and to a new synthesis approach based on these groups in Section V.

Section III develops a new and unified algorithm to extract prime implicants which, as a by-product, also detects disjunctive decomposition of a switching function. Implicants of both the function and its complement are detected simultaneously, and the algorithm is not sensitive to functional complexity. Its computational burden grows as \( n^3 \), where \( n \) is the number of function arguments, and its storage requirements grow as \( 2^n \) rather than \( 3^n \) (prime implicants need not be stored).

Section IV introduces a new class of transformation groups called the restricted affine groups (RAG). The elements of these groups are called prototype transformations, and they operate on the Cartesian product (or direct sum) of the domain \( \mathbb{Z}^n \) and range \( \mathbb{Z} \) of a switching function, where \( \mathbb{Z} = \{0, 1\} \), the two-element Galois field. They comprise the largest subgroup of the \((n + 1)\)-dimensional affine group which does not introduce feedback from the function output to its input. The RAG includes as subgroups all the transformation groups previously considered in the literature of switching theory. The prototype equivalence classes into which RAG partitions the space \( \mathcal{F} \) of all 2-valued functions on \( \mathbb{Z}^n \) are unions of the equivalence classes induced by its subgroups.

Section IV,A places RAG within the context of its classical subgroups, describes prototype transformations in mathematical and engineering terms,
and establishes a useful connection between linear and affine representations of group elements. By means of this connection, rational canonical forms for prototype transformations and methods for computing the parameters required in Polya's counting theorem were developed and applied by Lechner (1963) to show that the set of $2^{32}$ truth tables of 5-argument functions are grouped into exactly 48 prototype equivalence classes by the RAG.

Section IV.B explains how this method differs from the method used by Harrison (1964) for direct product groups. This section also presents data on prototype classes for the 3- and 4-argument cases and class counts for functions of a given weight (number of points $x$ such that $f(x) = 1$) under linear and affine groups on the domain of $f$. Ninomiya (1958) first applied the invariant properties of Fourier transforms to the classification of switching functions under prototype transformations for $n \leq 4$ arguments.

Section IV.C gives results of a sample and search technique which identified unique representatives for 46 of the 48 prototype equivalence classes for 5-argument functions. Parameters which separate (uniquely characterize) the 46 known classes are also tabulated; these can easily be computed from the Fourier transform of an arbitrary 5-argument function. Resolution of the remaining ambiguity (identification of the last two classes) remains an open problem.

Section V returns to the problem of combinational logic synthesis armed now with the tools of abstract harmonic analysis. Section V.A motivates the new approach by considering the overall context of large-scale-integrated circuit technology.

Section V.B proposes a new structural model or framework for combinational logic synthesis which reduces to either the conventional macrocellular or the newer microcellular array approach for extreme cases of particular functions. This approach, called encoded input logic, applies the prototype transformation of Section IV to reduce the complexity of normal form realization. The invariant properties of Fourier transforms in Section II,C yield practical synthesis algorithms which select the encoding transformations.

Section V.C presents a realistic example, a 6-input function taken from a universal logic module (ULM) example of Elspas et al. (1967). This example made use of a preliminary version of the prime implicant extraction algorithm written for an interactive time-sharing terminal. A minimal 2-level normal form of this function required eleven 7- or 8-input AND gates with a total of 104 gate inputs. By imbedding another function from the same prototype equivalence class within a network of eight EXCLUSIVE OR gates, an encoded-input logic realization was achieved with only 53 gate inputs. Eight AND gates, each with four or five inputs, were required in addition to the eight 2-input EXCLUSIVE OR gates. Of course, there are many ways to obtain an implementation which is less costly than a minimal 2-level normal form as soon as the 2-level (speed-related) constraint is relaxed. Besides its
compatibility with macrocellular and microcellular array technology, the principal advantage of the encoded-input logic is that the new, apparently powerful, computationally effective, and easily understood techniques of abstract harmonic analysis can be brought to bear on the problem.

Section V,D considers fundamental problems and extensions that have been motivated by the research presented in this chapter. These include synthesis of encoded-input threshold logic, probability distributions for spectral coefficients, canonical forms of prototype transformations on the direct sum $\mathbb{Z}^n + \mathbb{Z}^n$ and their application to multiple-output and many-valued functions, state assignments for sequential circuits, and further development of harmonic analysis along lines suggested by topological dynamics.

In conclusion, the many-faceted literature of combinational switching theory justifies a continuing search for new conceptual insights, unifying principles, and quantitative properties for combinational logic functions. In this respect, abstract harmonic analysis appears to make a unique contribution.

II. SURVEY OF ABSTRACT HARMONIC ANALYSIS

The plan of this section is as follows: Section II,A formally defines the abstract Fourier transform of a combinational logic function, then develops generalizations and computational techniques. It also provides historical background on applications to both switching and coding theory and analogies to other engineering applications of related transform techniques. Section II,B surveys the classical properties of the Fourier transform representation including the convolution theorem which is basic to the combinatorial applications of Section III. Section II,C develops the invariant properties of Fourier transforms under affine operators. These will be used to analyze the prototype equivalence relation in Section IV and to synthesize logic with linearly encoded inputs in Section V. Since the Fourier transform possesses unique advantages as an analytical tool for further research on the structure of switching functions, notice is given to computational techniques that might be used in each section.

A. THE FOURIER TRANSFORM OF A FUNCTION ON $\mathbb{Z}^n$

1. Notation

In this chapter $\mathbb{Z}$ will denote the 2-element field $\{0, 1\}$, $\mathbb{Z}^n$ will denote the $n$-dimensional vector space over the field $\mathbb{Z}$, and $\mathcal{F}$ will denote the set of all $n$-input, single-output functions from $\mathbb{Z}^n$ into $\mathbb{Z}$. For $N > 2$ (and not necessarily a prime power), $\mathbb{Z}_N$ will identify the ring of residue classes mod $N$; i.e.,
\[ Z = I/2I, \text{ and } Z_n = I/NI \text{ where } I \text{ is the ring of all integers. For clarity, another } \\
\text{symbol } X \text{ will also be used to identify the set } Z^n \text{ of all binary } n\text{-tuples when it is the domain for the space } F. \text{ The general element } x \text{ of } X \text{ will be represented by the row vector } (x_1, \ldots, x_n); \text{ the symbol } x^t \text{ will denote the column vector which is the transpose of } x. \text{ To index elements of } X, \text{ we will use the natural ordering of } n\text{-tuples defined by the correspondence between integers } 0 \leq i < 2^n \text{ and their radix-two expansions or binary } n\text{-tuples } x(i). \text{ (In other words, if } x(i) = (x_1, x_2, \ldots, x_n), \text{ then } i = x_12^{n-1} + x_22^{n-2} + \cdots + x_n. \) \\

Elements of \( F \) will be denoted by letters \( f, g, h. \) To define \( f \) more explicitly, we use the graph or truth table \( \{(x, f(x)) : x \in X, f(x) \in Z\} \) (a subset of the Cartesian product or direct sum \( X + Z \)), and its \( 2^n \)-tuple vector equivalent \( \{f_i = f(x(i)), 0 \leq i < 2^n\}; \) we also use the standard sum or level set representation \( f^{-1}(1) = \{x : f(x) = 1\} \) or its integer subset equivalent \( \{i : f(x(i)) = 1\}. \) The logical complement of a binary variable \( x_i \) will be denoted \( \overline{x}_i. \) We will also use the overbar symbol to denote the functional complement of a binary-valued function \( f = 1 - f. \) The terms argument, input, and variable will be used interchangeably to mean one of the \( x_i. \)

The residue class, modulo two, of the integer \( i \) will be written \( |i|_2; |b| \text{ also denotes the weight or real sum } b_1 + b_2 + \cdots + b_n \) of a binary \( n\)-tuple. The symbol \( \oplus \) will denote vector addition mod 2.

The \( 2^n \)-tuple \( \{f_i, 0 \leq i < 2^n\} \) is in one: one correspondence with the disjunctive canonical form for \( f(x), \) an expansion of \( f \) with respect to the fundamental product, or minterm, basis of the space \( F. \) The minterm basis functions are defined for \( 0 \leq i < 2^n \) by

\[ p_i(x) = (x_1)^{i_1}(x_2)^{i_2} \cdots (x_n)^{i_n} \]

where \( (i_1, i_2, \ldots, i_n) \) is the radix-two expansion of \( i, \) and \( x^k = x \) or \( \overline{x} \) if \( k = 1 \) or 0, respectively. The vector representation of \( p_i \) is

\[ p_i(j) = p_{ij} = \begin{cases} 1 & \text{iff } i = j \\ 0 & \text{otherwise} \end{cases} \]

In other words, \( p_i \) is the \( i \)th unit vector \( 0 \leq i < 2^n. \) Thus, instead of \( f(x) = \sum f_ip_i(x) \in F, \) we may write \( f = \sum f_ip_i \in Z^{2^n}. \) A similar correspondence between functions and vectors will be used when \( f \) is represented with respect to the Fourier transform basis.

2. **Definition of the Fourier Transform of a Function on \( Z^n \)**

The representation of \( f \) by its truth table, binary \( 2^n \)-tuple, or disjunctive canonical form is of value for three reasons: (1) it is a unique representation, (2) it explicitly identifies the value of \( f(x) \) for each state or configuration \( x \)
of its arguments, and (3) its tabular form provides a convenient starting point from which to derive simplified logic formulas for use in computer programs or logic circuit design. The abstract Fourier transform of $f$ is also a unique representation. Although it does not have advantage (2) this is more than compensated by its other unique advantages that will be demonstrated in this chapter.

The space $\mathcal{F}$ of all two-valued functions on $\mathbb{Z}^n$ has a locally compact Abelian or commutative group for its domain, and its range elements 0 and 1 can be added and multiplied as complex numbers. These two simple requirements are both necessary and sufficient to bring the methods of abstract harmonic analysis to bear on the study of this function space. In other words, an orthogonal basis set of Fourier transform kernel functions can be constructed for $\mathcal{F}$. A real (integer) valued function on $\mathbb{Z}^n$ will have a real (integer) valued transform.

The kernel or basis functions of the Fourier transform pair are defined in terms of a particular type of mapping from the $n$-dimensional vector space $\mathbb{Z}^n$ to the direct product of $n$ copies of the multiplicative subgroup $\{\pm 1\}$ on the unit circle of the complex plane. This mapping is such that the group sum of any two domain elements is mapped into the complex product of the images (i.e., the mapping is a group homomorphism, and group addition becomes complex multiplication). To motivate this definition, consider for example the cyclic group of integers modulo $N$. If the integers $x = 0, 1, 2, \ldots, N - 1$ are mapped into the points $\exp(2\pi ix/N)$ on the unit circle, then this mapping preserves the group operation; that is, the image of $x \oplus y$ is $\exp[2\pi i(x + y)/N]$ which is identical to the product $\exp(2\pi ix/N) \cdot \exp(2\pi iy/N)$. In other words, multiplication of points on the unit circle can be carried out by adding their “angles” or exponents.

There are actually $N$ group homomorphisms (rather than one) from the ring of integers modulo $N$ onto the unit circle; they are defined by

$$Q_k(x) = \exp(2\pi ikx/N) \quad (1)$$

In the case of functions on $\mathbb{Z}$, $N$ becomes 2 and the $N$ equally spaced points on the unit circle (one of which must be the multiplicative identity) become two real integers: $Q_k(x) = \exp(2\pi ikx/2) = (-1)^{kx} = \pm 1$ ($k = 0$ or 1). This mapping must be extended to the direct sum of $n$ 1-dimensional subspaces over $\mathbb{Z}$ (more generally, to the direct sum of $n$ cyclic Abelian groups). Define the image of $(x_1, x_2, \ldots, x_n)$ under the homomorphism $Q_{k_1, k_2, \ldots, k_n}(x) = Q_k(x)$ to be $(-1)^{k_1x_1}(-1)^{k_2x_2} \cdots (-1)^{k_nx_n}$ or $(-1)^{kx}$ where $k$ is the vector $(k_1, k_2, \ldots, k_n)$ which indexes the homomorphism. These functions are called “group characters” or Fourier transform kernel functions (Littlewood, 1940). From now on we will use $w$ instead of $k$ to index these functions (by analogy with engineering use of the Greek letter $\omega$ for complex frequency).
Example 1. For the case \( n = 2 \), a geometric as well as numerical derivation of the transform pair can be illustrated. (Arguments \( x_0, x_1 \) will be used instead of \( x_1, x_2 \) in this example.) The space \( \mathcal{F} \) consists of \( 2^4 = 16 \) different functions of 2 arguments \((x_0, x_1)\). To each integer between 0 and 15 (used as a function index), there corresponds a unique binary 4-tuple (the truth table for the function). The components of this 4-tuple are \( f_0 = f(0, 0), f_1 = f(0, 1), f_2 = f(1, 0), \) and \( f_3 = f(1, 1) \), respectively. For example, the function \( f^7 = \overline{x}_0 \lor \overline{x}_1 \) has the truth table \((1, 1, 1, 0)\). When its components \( f_i \) are written in ascending order from right to left \((f_3, f_2, f_1, f_0)\), this 4-tuple is also the binary code for its index, 7.

In Fig. 1a, these sixteen 4-tuples are represented as vertices of a 4-dimen-

\[ f^7 = \overline{x}_0 \lor \overline{x}_1 \]

\[ f^3 = \overline{x}_0 \land x_1 \]

\[ f^1 = x_0 \land x_1 \]

\[ f^0 = x_0 \land x_1 \]

\[ f^1 = x_0 \land x_1 \]

\[ f^2 = x_0 \lor x_1 \]

\[ f^3 = x_0 \lor x_1 \]

\[ f^4 = x_0 \land x_1 \]

\[ f^5 = x_0 \land x_1 \]

\[ f^6 = x_0 \land x_1 \]

\[ f^7 = x_0 \land x_1 \]

FIG. 1. The \( Q \)-basis for functions of two variables. (a) Representation as vertices on a 4-dimensional unit cube. (b) Negatives of the new basis vectors after the transformation \( y \rightarrow Q_x(x) \).

sional unit cube, with coordinate values 0 or 1. Formulas for some of the indexed functions are shown next to the corresponding vertices. Four unit basis vectors are also shown on Fig. 1a. These vectors are associated with the "minterm" functions \( f(x) = x_1 x_2, \overline{x}_1 \overline{x}_2, \overline{x}_1 x_2, \) and \( x_1 \overline{x}_2 \), respectively. The expansion of any function (vector) with respect to this basis is called its disjunctive canonical form. For example, the function \( \overline{x}_0 \lor \overline{x}_1 = \overline{x}_1 x_2 + x_1 \overline{x}_2 + \overline{x}_1 \overline{x}_2 \) will be evaluated in two steps to illustrate the geometric correspondence shown on the figure: (1) For each \( w \), evaluate \( y = xw^t \) (mod 2) for each \( x \). (2) Apply the transformation \( y \rightarrow (-1)^y = 1 - 2y \) to each result of step (1).

The following table shows \( y_w \) and \( Q_w \) for each \( w \) and \( x \):

\[ \begin{array}{ccc}
\hline
w & x & y_w \\
\hline
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\hline
\end{array} \]
\[
y_w(x) = x w^i \pmod{2}
\]

\[
Q_w(x) = 1 - 2y_w(x)
\]

<table>
<thead>
<tr>
<th>( w )</th>
<th>0 0</th>
<th>0 1</th>
<th>1 0</th>
<th>1 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
Q_0 = 1\text{-}1\text{-}1\text{-}1 = Q_1
\]

The rows defining \( y_w(x) \) are mod 2 linear functions of \( x \) and correspond to vertices numbered 0, 10, 12, and 6 on the left side of the figure. The transformation \( y \rightarrow Q_w(x) \) represents a doubling in size and change of sign in all coordinates of the unit cube, followed by a translation of origin from one vertex to the center of the cube. Negatives of the new basis vectors are shown on Fig. 1b so their endpoints terminate on vertices of the corresponding linear functions \( xw^i \). The new basis vectors are also orthogonal (as will be shown rigorously later).

**Example 2.** Representation of the function \( \bar{x}_0 \vee \bar{x}_1 \): Using ordinary Euclidean geometry, we can obtain the projections of the vector \( f = (1, 1, 1, 0) \), which represents \( f_i(x) = \bar{x}_0 \vee \bar{x}_1 \) onto each row of the \( \textbf{Q} \) matrix (see Section II, A, 4):

\[
f^* = f \cdot \textbf{Q} = (1, 1, 1, 0) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} = (3, 1, 1, -1)
\]

(2)

Since \( \textbf{Q}^{-1} = 2^{-*} \textbf{Q} \) (as will be shown later), \( f = f^* \textbf{Q}^{-1} = 2^{-*} f^* \textbf{Q} \). This means that \( f(x) \) has a unique expansion of the type \( f(x) = \alpha \sum f_i^* \textbf{Q}_i(x) \); in this example, \( f(x) = \alpha (3 \textbf{Q}_0(x) + \textbf{Q}_1(x) + \textbf{Q}_2(x) - \textbf{Q}_3(x)) \) where \( \alpha = \frac{1}{4} \) is a normalizing factor so that \( f = 0 \) or 1.

**Theorem 2.1.** The set of Fourier kernel or basis functions \( \{ \textbf{Q}_w(x) = (-1)^{wx^i}, w \in \mathbb{Z}^n \} \) (also known as the character group of \( \mathbb{Z}^n \)) is an orthogonal basis for the space of all complex-valued functions on \( \mathbb{Z}^n \).

**Proof:** The \( \textbf{Q}_w \) are all distinct; for suppose \( \textbf{Q}_u = \textbf{Q}_v \), although \( u \) and \( v \) differ in (say) the \( i \)th coordinate. Then \( u_i \neq v_i \) and \( \textbf{Q}_u(e_i) \neq \textbf{Q}_v(e_i) \) when \( x \) is the \( i \)th unit vector \( e_i \). Any set of \( 2^n \) distinct kernel functions \( \textbf{Q}_w(x) \) will span
(be a basis) if they are orthogonal. To prove orthogonality, expand the inner product:

\[ (Q_w, Q_x) = \sum_x (-1)^{w_x}(-1)^{x_x} = \sum_x \prod_j (-1)^{(w_j + x_j)x_j} = \sum_x (-1)^{(w + x)x} \]  

(3)

Now \((w_j + y_j)\) has the range 0, 1, and 2 in each exponent; however, because \((-1)^{2x_j} = (-1)^0 = 1\) regardless of the value of \(x\), \(w_j + y_j\) can be replaced by a modulo two sum for each \(j\). Defining \(u = w + y \mod 2 = w \oplus y\),

\[ (Q_w, Q_x) = \sum_x (-1)^{ux} \]  

(4)

where the sum is over all of \(X\).

If \(w = y\), then \(u = 0\) and every term of this sum is 1, which implies \((Q_w, Q_w) = 2^n\). If \(w \neq y\), then \(u \neq 0\) and the inner product \(ux\) is a projection of \(X\) into a 1-dimensional subspace, or 2-element subgroup. Its kernel, the set \(K = \{ x \in X : ux = 0 \}\), is a subspace of dimension \(n - 1\), and \(X - K\) is a single coset \(C = \{ x \in X : ux = 1 \}\). Splitting the sum into two parts, since \(K\) and \(C\) are both \(2^{n-1}\)-element subsets of \(X\), we have

\[ (Q_w, Q_x) = \sum_{x \in K} (-1)^{ux} + \sum_{x \in C} (-1)^{ux} = \sum_{x \in K} (-1)^0 + \sum_{x \in C} (-1)^1 = 0 \]  

(5)

which proves orthogonality.

**DEFINITION.** The Abstract Fourier Transform of \(f\) is the integer-valued function \(f^*\) which defines (up to a scale factor) the expansion coefficients or coordinates of \(f\) with respect to the basis \(\{Q_w(x) = (-1)^{w_x} ; w \in \mathbb{Z}^n\}\). The domain of definition of \(f^*\) is another vector space \(W\) of binary \(n\)-tuples isomorphic to \(X\). For notational economy, the \(j\)th element \(w(j)\) of \(W\) will be defined as the binary code for \(j\), \(f^*(w(j))\) will be denoted \(f_j^*\), and \(Q_j^*(x(i))\) or merely \(Q_{ji}\) will denote \(Q_{w(j)}(x(i))\). Then the Fourier expansion of \(f\) can be represented two ways:

\[ f(x) = 2^{-n} \sum_w f^*(w)(-1)^{w_x} = 2^{-n} \sum_w f^*(w)Q_w(x) \]  

(6)

or

\[ f_i = f(x(i)) = 2^{-n} \sum_j f_j^*Q_j(x(i)) = 2^{-n} \sum_j f_j^*Q_{ji} \]  

(7)

**Example 3.** Take \(n = 2\), \(f^*\) and \(Q\) from the preceding example. Then

\[ f = 2^{-2}f^*Q \]  

becomes \((f_0, f_1, f_2, f_3) = 2^{-2}(3, 1, 1, -1)Q = (1, 1, 1, 0)\) which is the truth table for the function \(f(x) = x_0 \lor \overline{x}_1\).
3. Derivation of the Transform Pair

To derive the transform coefficients \( f_j^* \), simply assume the existence of the Fourier series expansion \( f(x) = 2^{-n} \sum_{j} f_j^* Q_j(x) \), multiply both sides by \( Q_k(x) \), sum over \( X \), and make use of the orthogonality property \( (Q_j, Q_k) = 2^k \delta_{jk} \) of the basis functions \( (\delta_{jj} = 1; \delta_{jk} = 0 \text{ for } j \neq k) \).

\[
\sum_x f(x) Q_k(x) = 2^{-n} \sum_{j=0}^{2^n-1} f_j^* \sum_x Q_j(x) Q_k(x) = 2^{-n} \sum_j f_j^* (2^k \delta_{jk}) = f_k^*
\]

(8)

**DEFINITION.** Using the above notation, the abstract Fourier transform pair is defined for any (real-valued) function \( f \) in \( \mathcal{F} \) as follows (using real arithmetic):

\[
f_j^* = f^*(w(j)) = \sum_x f(x) Q_j(x) = \sum_i f_i Q_{ij}
\]

(9)

\[
f_i = f(x(i)) = 2^{-n} \sum_j f_j^* Q_j(x(i)) = 2^{-n} \sum_j f_j^* Q_{ji}
\]

(10)

**Example 4.** Collecting the preceding examples together produces the following transform pair for the function \( f^T = \bar{x}_0 \lor \bar{x}_1 \):

\[
(f_0^*, f_1^*, f_2^*, f_3^*) = (1, 1, 1, 0)Q = (3, 1, 1, -1)
\]

\[
(f_0, f_1, f_2, f_3) = 2^{-2}(3, 1, 1, -1)Q = (1, 1, 1, 0)
\]

where \( Q \) is the matrix of Example 2.

Since the domains of \( f \) and \( f^* \) are isomorphic, and the transform is symmetric, the same * superscript may be used to denote the inverse transform which maps \( f^* \) back into \( f : f = 2^{-n}(f^*)^* \). The scale factor \( 2^{-n} \) which is required to normalize the transform pair is included in the inverse transform to make \( f^* \) an integer-valued function. It is explicitly written so that no ambiguity exists as to whether the * superscript denotes a forward or reverse transform.

The matrix formulation of this transform pair is convenient for computation. The integer-valued \( 2^n \)-tuples (row vectors) \( f = \{f_j, 0 \leq j < 2^n\} \) and \( f^* = \{f_i^*, 0 \leq i < 2^n\} \) are related by the symmetric transform matrix \( Q = \{q_{ij}\} \) as follows:

\[
f^* = fQ; \quad f = f^*Q^{-1} = 2^{-n}f^*Q
\]

(11)
The direct analogy between this transform pair and the classical $n$-dimensional Fourier series representation of a function of period $p$ in each of its coordinates is apparent if we use $p$ instead of 2 for the characteristic of the finite field over which $X$ and $W$ are defined. In this case, the abstract transform pair becomes

\[
    f^*(w) = \sum_x f(x) \exp(2\pi i(wx^p)/p) \tag{12}
\]

\[
    f(x) = p^{-n} \sum_w f^*(w) \exp(-2\pi i(wx^p)/p) \tag{13}
\]

Note that the basis functions take $p$ equally spaced values on the unit circle. For $p = 2$, these values are $\pm 1$. For all other $p$, they become complex-valued. For a finite domain, both the function and its transform have finite and discrete domain of definition. This is a major difference from the conventional Fourier transform pair for a function of $n$ variables, each with period $p$. The latter has a continuous finite domain $X$ and a discrete countably infinite transform domain $W$. In this case $W$ includes all $n$-tuples of integers.

\[
    f^*(w) = \int_0^p \cdots \int_0^p f(x) \exp(2\pi i(wx^p)/p) \, dx_1, \ldots, dx_n, \quad w \in \mathbb{Z}^n \tag{14}
\]

\[
    f(x) = p^{-n} \sum_{w \in \mathbb{Z}^n} f^*(w) \exp(-2\pi i(wx^p)/p), \quad x \in [0, p]^n \tag{15}
\]

4. A Fast Fourier Transform Algorithm

From Eq. (11) computation of $f^*$ from $f$ (or vice versa) apparently requires $2^n$ dot products, each involving $2^n - 1$ additions or subtractions. Ninomiya (1958) computed a slight variation of $f^*$ by summing $m$ basis vectors, where $m$ is the number of nonzero components of $f$. Golomb (1959) overlaid a set of $2^n$ templates on an array representing $f$ to compute a set of $2^n$ invariants completely equivalent to (but not as symmetrically defined as) $f^*$. Each template selected $2^{n-1}$ components of $f$ to be summed. The following computational algorithm evaluates $f^*$ or its inverse in no more than $n2^{n+1}$ integer-valued addition or subtraction operations (Lechner, 1963a,b). This algorithm has been programmed for a digital computer and requires only $2^n$ cells of working storage. The transform $f^*$ can be overlaid within the $2^n$ cells of storage used for the truth table of $f$ itself. This algorithm is a special case of the fast Fourier transform algorithm now in widespread use on digital computers (Good, 1958; Bergland and Hale, 1967).
The algorithm depends on the recursive definition of the matrix \( Q \), which is identical to the Hadamard transform matrix (Golomb et al., 1964). For example, if \( Q_n \) denotes the matrix \( Q \) of order \( 2^n \), then

\[
Q_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad Q_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}
\]

and in general, if \( Q^{[n]} \) denotes the \( n \)th Kronecker power of \( Q_1 \),

\[
Q_{n+1} = \begin{bmatrix} Q_n & Q_n \\ Q_n & -Q_n \end{bmatrix} = Q_1 \times Q_n = Q_1^{[n+1]}
\]

where \( A \times B \) denotes the Kronecker product of two matrices (Bellman, 1960). The following expansion theorem may be verified by direct multiplication:

\[
A^{[n]} = \prod_{k=0}^{n-1} (I^{[n-k-1]} \times A \times I^{[k]}) = \prod_{k=0}^{n-1} S^{[k]}
\]

where \( \prod \) denotes regular matrix multiplication, and \( I^{[k]} \) is the identity matrix of order \( 2^k \). Now let \( A = Q_1 \) and \( S^{[k]} = (I^{[n-k-1]} \times Q_1 \times I^{[k]}) \). Then

\[
f^* = f \cdot S^{[1]} \cdot S^{[2]} \cdots S^{[n]}
\]

If the multiplications are done from left to right, this reduces to a simple case of the well-known fast Fourier transform algorithm (Good, 1958; Bergland and Hale, 1967). The matrix \( Q \) (the order \( 2^n \) will be understood) has been factored into the ordinary matrix product of \( n \) factors \( S^{[k]} \) each of which is trivial and requires no storage of the \( a_{ij} \) coefficients. The \( k \)th factor involves \( 2^{n-k} \) submatrices of the form \( \begin{bmatrix} \vdots & \vdots \\ \vdots & \vdots \end{bmatrix} \) along its main diagonal; each submatrix forms the sum and difference of a pair of \( 2^{k-1} \)-tuples. Thus, each of the \( n \) factors requires exactly \( 2^{n+1} \) integer additions or subtractions, and the entire transform for a \( 2^n \)-dimensional vector \( f \) requires \( n2^{n+1} \) elementary operations.

**Example 5.** Figure 2 is a diagrammatic representation of the transform computation for a particular function of five arguments (from Table XI). A negative coefficient is indicated by an overbar. The last two rows of Fig. 2 permute the coefficients \( f_n^*(w) \) into ascending order of their weight \( |w| \) to show the lexicographic ordering of coefficients that makes this representative of prototype class 30B unique.
5. Range Translation to Improve Symmetry

The range of $f$ up to now has been restricted to the integers 0 and 1. However, the Fourier expansion $f^*$ represents any (real-valued) function $f$ on $\mathbb{Z}^n$ as a (real) sum of $(\pm 1)$-valued kernel functions. The translation $f_N = \frac{1}{2} - f$ is a trivial modification of the range, but it produces a transform $f_N^*$ with greatly improved symmetry properties. We append the subscript $N$ because Ninomiya (1958) first derived this transform by reversing the sign of each component of $f^*$ and adding $2^{n-1}$ to the first component $f^*(0)$. The following theorem shows that Ninomiya’s transform is just the Fourier transform of $\left(\frac{1}{2} - f\right)$:

**Theorem 2.2.** Let $f_0(x) = \frac{1}{2} - f(x)$. Then $f_N^*(w) = 2^{n-1} \delta_{w_0} - f^*(w)$ for all $w$ in $\mathbb{Z}^n$.

**Proof:** Let $f_N(x) = -f(x) + \frac{1}{2}$. The linearity properties of the Fourier transform (as an expansion with respect to orthogonal basis functions in real Euclidean $2^n$-space) imply that $(f + g)^* = f^* + g^*$. Therefore,

$$f_N^* = -(f)^* + \left(\frac{1}{2}\right)^* = -(f)^* + (\frac{1}{2})^{(1)^*}$$
The transform of a constant \((1)\) has \(j\)th coordinate \(\sum w \cdot (-1)^{w \cdot j}; w = w(j)\). In the proof of orthogonality of the Fourier basis functions, we observed that this expression reduces to \(2^n \cdot 0\). Therefore, \(f_N^* = -f_j^* + 2^{n-1} \cdot 0\).

In other words, the \(j\)th coefficient \(f_N^*(w(j))\) of Ninomiya’s coordinate representation for \(f\) is equal to \(f_j^*\) with its sign reversed for \(0 < j < 2^n\), and \(f_N^*(0) = 2^{n-1} - f_0^*\) for \(j = 0\).

One way in which \(f_N^*\) is more symmetric than \(f^*\) is that mod 2 complementation of \(f\) produces complete sign reversal of all coefficients of \(f_N^*\), but not of \(f^*\). Using real addition, the binary complement of \(f\) becomes

\[
\bar{f}(x) = 1 - f(x)
\]

and its Fourier transform \((\bar{f})^* = (1)^* - f^* = 2^n \cdot 0 - f^*(w)\). In other words, \(2^n\) is added to the first component of \(f^*\) after sign reversal in each coordinate. On the other hand, \((\bar{f})_N = \frac{1}{2} - f = \frac{1}{2} - (1 - f) = \frac{(1 - f)}{2} = (-f_N)\). Therefore, \((\bar{f})_N^* = -f_N^*\) (i.e., the only effect of functional complementation is to multiply the entire transform \(f_N^*\) by \((-1)\)). This symmetry property simplifies testing for equivalence of two functions under transformation groups which include functional complementation.

6. Relation to EXCLUSIVE OR Canonical Forms

Switching functions are often defined more compactly by algebraic expressions other than their disjunctive canonical forms. For example, \(f(x)\) can be expressed as a sum of partial products of the variables. The variables may be complemented or uncomplemented independently from term to term. When the summation is Boolean, such forms are known as normal forms; when the summation is mod 2, they are known as (consistent or inconsistent) \(\Delta\)-forms (Calingaert, 1961). By repeated use of the three identities below, such forms may be expressed as real sums of uncomplemented partial or reduced product functions \(r_j(x)\) defined by

\[
r_j(x) = \prod_{i=0}^{n-1} (x_i)^{b(i)}
\]

where \(b(i)\) is the \(i\)th component of the radix-two expansion of \(i\), \(x^0 = 1\), and \(x^1 = x\). The required identities are (using \(\vee\) for Boolean addition and \(\Delta\), the equivalent of \(\oplus\), for mod 2 addition):

\[
\bar{x} = 1 - x
\]
\[
x \Delta y = x + y - 2xy
\]
\[
x \vee y = x + y - xy
\]
Example 6. (Real expansion with respect to reduced product functions)

\[ f(x) = x_0 \bar{x}_1 \Delta x_2 \]
\[ = x_0 \bar{x}_1 + x_2 - 2(x_0 \bar{x}_1 x_2) \]
\[ = x_0 - x_0 x_1 + x_2 - 2x_0 x_2 + 2x_0 x_1 x_2 \]
\[ = x_1 - x_0 - 2x_0 x_2 - x_0 x_1 + 2x_0 x_1 x_2 \]
\[ = r_1 + r_2 - 2r_3 - r_6 + 2r_7 \]

which may be written as

\[ f(x) = \sum_j g_j r_j(x) = rg^t \]  \hspace{1cm} (21)

where

\[ g = (0, 1, 0, 0, 1, -2, -1, 2) \]

and

\[ r = (r_0(x), r_1(x), \ldots, r_7(x)) \]

The Fourier expansion of \( f(x) \) will now be obtained in terms of a general linear transformation relating the functions \( r_j(x) \) to the Fourier basis set. The results will incidentally define the vector \( g \) and prove that the set of functions \( \{r_j(x), 0 \leq j < 2^n\} \) is linearly independent over the reals, hence is a basis (although not orthogonal); it follows then that the expansion of \( f(x) \) with respect to the \( r_j \) basis over the real field is also unique. By definition,

\[ Q_4(x) = (-1)^{x w^t} = \prod_{j=1}^n (-1)^{x w_j} = \prod_{j=1}^n (1 - 2x_j w_j(k)) \]  \hspace{1cm} (22)

If the last expression is expanded as a real sum of partial products of (un-complemented) variables, only those \( r_j \) will appear which are factors of \( r_k \); these will have as coefficients the integers \((-2)^i\) to the power \( \sum w_j(j) = |w(j)| \).

Now \( r_j \) is a factor of \( r_k \) if and only if \( w(j) \leq w(k) \) (i.e., \( w(j) \leq w_i(k) \) for \( 1 \leq i \leq n \)). Calingaert (1961) has shown that \( w(j) \leq w(k) \) if and only if the binomial coefficient \( |j| \) has odd parity. The expression for \( Q_4(x) \) may therefore be written as

\[ Q_4(x) = \sum_{j=0}^{2^n-1} \left[ \binom{k}{j} \right] (2)^{|w(j)|} r_j(x) \]  \hspace{1cm} (23)

Calingaert (1961) defined \( A \) as the \( 2^n \) by \( 2^n \) matrix having \( |j| \) as its \((k,j)\)th element. The diagonal matrix having \((-2)^{|w(j)|}, 0 \leq j < 2^n \) as its \( j \)th diagonal element is denoted by \( W \). In matrix form, \( q(x) = r(x) WA^t \) where

\footnote{Here \(|j|\) denotes the residue, mod 2, of the binomial coefficient.}
the kth component of q or r is the function $Q_k(x)$ or $r_k(x)$, respectively, and the equation is an identity in the variables (i.e., true for each configuration of x).

The disjunctive canonical form of $r_j(x)$ is $r_j(x) = \sum b_{ji} p_i(x)$ where $b_{ji} = 1$ if and only if $w(i) \leq w(j)$. Hence, $b_{ji} = |ij|_2$ which is the $(j, i)$th element of $A$. Therefore

$$r_j(x) = \sum_i |i\rangle |j\rangle_2 p_i(x)$$

(24)

or in matrix form,

$$r(x) = p(x)A$$

where the jth column of $A$ is the vector representation of the disjunctive canonical form of $r_j(x)$. Now $A$ is triangular with determinant 1 since $|i\rangle = 1$ and $|j\rangle = 0$ for $j > i$. Therefore, the rows of $A$ are linearly independent and its inverse exists:

$$p(x) = r(x)A^{-1}$$

(25)

Furthermore, for every configuration of the variables $x$,

$$Q(x) = r(x)WA^t = p(x)AWA^t$$

(26)

$$f^*(w) = f(x)Q(x)$$

(27)

The above equation is an identity between the ith columns of $AWA^t$ and $Q$, which proves the following theorem:

**THEOREM 2.3.**

$$Q = AWA^t$$

(over the reals)

(28)

where

$$(A)_{ij} = |i\rangle |j\rangle_2$$

and

$$(W)_{ij} = (-2)^{|w(i)|} \delta_{ij}$$

The linear transformations relating the bases $r(x)$, $p(x)$, and $q(x)$, or the coefficient vectors $g, f$, and $f^*$ of the expansion of an arbitrary binary (or real) valued function with respect to these bases, are shown diagrammatically below in Fig. 3 (see Lechner, 1963a,b).

The transformation $f = gA^t$ (over the reals) is similar to the equation
\[ f = hA' \text{ (over } \mathbb{Z}_2 \text{)} \] which relates \( f \) to the expansion coefficients \( h \) of Calingeart's reduced product or \( \Delta \)-sum canonical form (over \( \mathbb{Z}_2 \)). Over \( \mathbb{Z}_2 \), \( A \) is self-inverse, so \( h = fA' \text{ (mod } 2) \); over the reals, however, \( A \neq A' \)

**Example 7.** The Fourier transform \( f^* \) will be derived from the \( \Delta \)-form expansion vector \( g = (0, 1, 0, 0, 1, -2, -1, 2) \) of the preceding example of this section. From the diagram above, the appropriate transformation is

\[ 2^{-n} \cdot f^* = (gW^{-1})A^{-1} = yA^{-1} = (A')^{-1}y^t \quad (29) \]

The final product is expanded below (\( y \) having been precomputed by multiplying corresponding components of \( g \) and \( W^{-1} \)).

\[
\begin{align*}
2^{-n} \cdot f^* &= \begin{bmatrix}
1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\
0 & 1 & 0 & -1 & 0 & -1 & 0 & 1 \\
0 & 0 & 1 & -1 & 0 & 0 & -1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 1 & -1 & -1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix} \begin{bmatrix}
0 \\
-\frac{1}{2} \\
0 \\
0 \\
-\frac{1}{2} \\
-\frac{1}{2} \\
0 \\
-\frac{1}{2} \\
\end{bmatrix} \\
= \begin{bmatrix}
\frac{1}{2} \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix} \\
\end{align*}
\]

(30)

The last column checks with the result \( f^* = (4, -2, 0, 2, 0, -2, 0, -2) \) obtained by the direct transformation \( f^* = fQ \).

### 7. Historical Note

Muller (1954) was apparently the first to use this transformation on switching functions. He derived the Fourier kernel functions and classified all
4-argument truth tables eight ways according to their empirically-determined expansion coefficients with respect to this basis. The Fourier basis vectors are closely related to the Reed–Muller class of error-correcting codes (Reed, 1954). Muller’s pioneering work inspired Ninomiya to undertake a more analytic study of the properties of the Fourier expansion (without explicitly characterizing it as such). Ninomiya formalized Muller’s concept of functional equivalence and determined many of its properties. Both Muller and Ninomiya were apparently unaware of the unifying principle which the restricted affine group over \( \mathbb{Z}_2 \) (see Section IV) provides for their notion of equivalence.

This author’s interest in harmonic analysis was motivated chiefly by the extensive thesis research of Ninomiya (1958). Additional inspiration was drawn from earlier applications of Fourier analysis to coding theory, particularly by Zierler (1960) and Wells (1960). Berlekamp, in a private communication (1969) indicated that Ninomiya’s prototype equivalence relation is now being applied to coding theory by Berlekamp et al. who have identified all 48 prototype classes for \( n = 5 \) (see Section IV,C). Menger (1969) considered a different type of transform for function spaces whose range was not a subset of the complex field, but the finite field \( GF(p) \). The classical properties of Fourier transforms are not valid, and the utility of Menger’s transform has yet to be determined.

Recent applications of the Rademacher–Walsh transform to communication theory have aroused much interest among engineers. This transform is a permutation of the Hadamard transform applied to a \( 2^n \) samples of a real-valued function (Whelchel et al., 1968). If the sampled function is regarded as a function of the binary \( n \)-tuples which are radix-two expansions for the indices of their sample points, the theory developed in this chapter is applicable. However, our point of view is only relevant for functions with a discrete (e.g., 2-valued) range, because partitioning the function domain into level sets is essential to further theoretical development (see Section V,D).

**B. SURVEY OF CLASSICAL PROPERTIES**

This section proves three important properties of Fourier transforms for the domain \( \mathbb{Z}^n \). Other miscellaneous properties are also mentioned herein. Discussion of the invariant properties of Fourier transforms under affine operators will be deferred until Section IV,C. The three major theorems presented herein are the Convolution theorem, the Quotient Group Character theorem, and the Poisson Summation theorem. Their proofs are simple consequences of the transform definition for the domain \( \mathbb{Z}^n \) and could easily be extended to a direct sum of arbitrary cyclic groups (every finite Abelian group is isomorphic to one of this type). However, direct proofs for \( \mathbb{Z}^n \) should
expedite wider use of these new computational tools for combinational logic analysis. Most of this material was adapted from the text by Loomis (1953). Several recent books contain more extensive discussions of group theoretic applications (Rudin, 1962; Hewitt et al., 1963).

1. The Convolution Theorem

The convolution theorem is one of the most powerful tools of linear functional analysis. In 1963 the author completed an (unpublished) survey which interpreted abstract harmonic analysis in the context of switching theory. The material in this section is based on that survey. Applications of the convolution theorem were elusive until 1969 when the algorithm described in Section III was discovered. The convolution theorem for functions on \( \mathbb{Z}^n \) can be generalized to arbitrary finite Abelian groups (direct sums of cyclic groups, or mixed radix number representations) in a straightforward way.

**DEFINITION (Convolution Sum).** Let \( X \) be an \( n \)-dimensional vector space over the 2-element field. The convolution sum \( f * g \) for any given pair of real- (or integer-) valued functions \( f \) and \( g \) on \( X \) is a real- (or integer-) valued function defined at every point \( c \) of \( X \) by the following sum over all elements of \( X \):

\[
(f * g)_c = \sum_{x \in X} f(x \oplus c)g(x)
\]

(31)

Under certain summability conditions (always satisfied for finite domains), the convolution sum of a real- or complex-valued function defined on an Abelian group is identical to the inverse Fourier transform of the componentwise product of their Fourier transforms. Symbolically, \( f * g = (f^* \cdot g^*)^* \). For any finite domain, this convolution theorem is a simple consequence of the transform definition. In ordinary Fourier analysis, this property eliminates a large fraction of the multiplications required for a convolution sum. In the direct-sum approach, this number grows as the square of the number of the points for which \( f \) and \( g \) both have significant nonzero values. Computational advantages of this theorem also occur for finite groups like \( \mathbb{Z}_2^n \), but for completely different reasons.

We will state and prove this theorem for the domain \( \mathbb{Z}^n \) using transform notation of the previous section.

**THEOREM 2.4 (Convolution Theorem).** In the notation of the preceding sections, let \( f \) and \( g \) be any two integer, real- or complex-valued functions on \( \mathbb{Z}^n \), let \((f * g)_c \) denote their convolution sum evaluated at \( x = c \), and let
(f* g*) denote the componentwise product \( f^*(w)g^*(w) \) of their respective Fourier transforms \( f^* \) and \( g^* \). Then, for each \( c \) in \( \mathbb{Z}^n \), \( (f^* * g)^* c = 2^{-n}(f^* g^*)^c \) evaluated at the point \( x = c \).

**Proof:** Take the product of the two transform definitions
\[
(f^* g^*)_w = \sum_x (\omega^*)^{x} f(x) \sum_y (\omega)^{y} g(y) = \sum_{x, y} (\omega^{x+y}) f(x) g(y)
\]
(32)

Now let \( y = x \oplus c \), sum over \( x \) with \( c \) fixed, then sum over \( c \):
\[
(f^* g^*)_w = \sum_c (\omega^*)^{c} \sum_x f(x) g(x \oplus c) = \sum_c (\omega^*)^{c} (f * g)_c = (f * g)_w^*
\]
(33)

The final summation is merely the Fourier transform of \( f * g \), evaluated at \( w \). Transposing sides of this identity in \( w \) and taking the inverse transform on both sides produces the following result which is an identity for all \( c \) in \( \mathbb{Z}^n \):
\[
(f * g)_c = 2^{-n} \sum_w (\omega^*)^{w} (f^* g^*)_w = (f * g^*)_c
\]
(34)

**Example 8.** Let \( f = (1, 1, 1, 0) \) and \( g = (0, 1, 1, 1) \); \( (f * g) \) is calculated directly below:

<table>
<thead>
<tr>
<th>( x )</th>
<th>( f(x) )</th>
<th>( g(x \oplus c) ) ( c = (0, 0) )</th>
<th>( c = (0, 1) )</th>
<th>( c = (1, 0) )</th>
<th>( c = (1, 1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(0, 1)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(1, 0)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(1, 1)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[
\sum f(x) g(x \oplus c) = 2 \quad 2 \quad 2 \quad 3
\]

The convolution theorem obtains the same result as follows:
\[
f^* = fQ = (1, 1, 1, 0)Q = (3, 1, 1, -1)
\]
(35)
\[
g^* = gQ = (0, 1, 1, 1)Q = (3, -1, -1, -1)
\]
(36)
\[
(f^* g^*) = (9, -1, -1, 1)
\]
Taking the inverse transform produces an identical result:

\[ 2^{-n}(f \ast g)Q = \frac{1}{2}(9, -1, -1, 1) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} = (2, 2, 2, 3) \]  \hspace{1cm} (37)

2. Quotient Group Character Theorem

Suppose \( V \) is a \( k \)-dimensional subspace of \( \mathbb{Z}^n \) (an additive subgroup of order \( 2^k \)). Then \( \mathbb{Z}^n \) has a disjoint decomposition into \( 2^{n-k} \) cosets \( V + c \), one of which (for \( c = 0 \)) is \( V \) itself. These cosets form a group under (mod 2) addition, denoted \( \mathbb{Z}^n/V \), called the quotient group of \( \mathbb{Z}^n \) (mod \( V \)). The theorem to be proved states that any function which is constant on each coset of \( V \) can be expanded in terms of a subset of \( 2^{n-k} \) Fourier basis functions called "quotient group characters," defined by \( QG(\mathbb{Z}^n/V) = \{Q_w(x) : Q_w(x) = 1 \text{ for all } x \in V \} \). This theorem is a classical result of group character theory (Littlewood, 1940). Loomis (1953) gives an abstract proof. For a specific type of subspace, a direct proof is given herein, to bring out the significance of this theorem for our later applications.

This theorem will be useful in computing the Fourier transform of any function which is known a priori to be constant on each coset of a subspace of \( V \). (In particular, the functions corresponding to \( k \)-cells or subcubes of \( \mathbb{Z}^n \) are of this form.)

A "degenerate" function is one that depends on fewer than \( n \) linear combinations of its arguments. A corollary of the theorem states that degenerate functions (and their arguments) can be identified by computing the rank of the set of vectors \( w \) corresponding to the nonzero Fourier coefficients \( f \ast(w) \).

**THEOREM 2.5 (Quotient Group Characters).** Let \( V \) be a subspace of \( \mathbb{Z}^n \). Then the subset of Fourier basis functions \( Q_V = \{Q_w(x) : xw^t = 0, \text{ for all } x \in V \} \) is a complete orthogonal basis for the subspace of \( \mathcal{E} \) consisting of all functions that are constant on cosets of \( V \). No other functions are generated by this basis. If \( V \) has dimension \( k \), then \( Q_V \) consists of just those \( 2^{n-k} \) functions \( Q_w(x) \) for which \( w \) is orthogonal to all \( x \in V \). If \( V \) has a basis consisting of \( k \) unit vectors, then \( Q_w \in Q_V \) iff \( w \) is in the space generated by that subset of \( n - k \) unit vectors which are not in the basis for \( V \). (In either case, the set of all vectors \( w \) such that \( w^t x = 0 \) for all \( x \) in \( V \) is called the complementary subspace or nullspace \( V' \) of \( V \).)
Proof: For every \( k \)-dimensional subspace \( V \subseteq \mathbb{Z}^n \), there is a (nonunique) set \( S \) of \( 2^n - k \) elements (called coset leaders) \( \{ y_i \in \mathbb{Z}^n : y_i + V \neq y_j + V \text{ unless } i = j \} \). Here \( y + V = \{ y \oplus z : z \in V \} \). Furthermore, every \( x \) in \( \mathbb{Z}^n \) has a unique representation as \( x = y \oplus z, y \in S \) and \( z \in V \). In terms of this decomposition, the Fourier transform of \( f \) is defined as follows:

\[
f^*(w) = f^*(w) = \sum_{z \in V} \sum_{y \in S} f(y \oplus z)(-1)^{wy}(-1)^{wz} \tag{38}
\]

If \( f \) is constant on cosets of \( V \), then \( f(y \oplus z) = f(y) \) for all \( z \) in \( V \) and any \( y \) in \( S \). Furthermore, if \( w = w_1 \oplus w_2 \) with \( w_1 \in V' \) and \( w_2 \in V \), then \( w_1z^t = 0 \), and dimension \( (V') + \) dimension \( (V) = n \). Therefore,

\[
f^*(w) = \left[ \sum_{z \in V} (-1)^{wz} \right] \left[ \sum_{z \in V} f(y)(-1)^{wy} \right] = (2^k \delta_{w_20})(f \mid_\Gamma S)_w^* \tag{39}
\]

The first term arises because \( z \rightarrow wz^t \) is a homomorphism from \( V \) into \( \mathbb{Z} \) for every \( w \) (see the proof of orthogonality in Section II.A.3). Therefore, the first sum is zero unless \( w \in V' \), \( 2^k \) for \( w \in V' \). The second term \( (f \mid_\Gamma S)^*_w \) denotes the Fourier transform of \( f \) as a function on the set \( S \) alone. The symbol \( \Gamma \) denotes the restriction of \( f \) to the subset \( S \) of \( \mathbb{Z}^n \). Since \( S \) has \( 2^n - k \) elements, the function \( f \) restricted to \( S \) is a vector of dimension \( 2^n - k \). Since \( \delta_{w20} = 0 \) unless \( w_2 = 0 \), the only nonzero Fourier coefficients \( f^*(w) \) in the transform of \( f \) are those for which \( w = w_2 \oplus w_1 = w_1 \) is in \( V' \). In other words \( f^* = 0 \) except on \( V' \).

So far, we have not used the condition that \( V \) has a basis consisting of unit vectors. In this case, \( S \) can be identified with \( V' \), because \( V' \) contains all the coset leaders of minimum weight.

To prove the converse (all functions generated by the orthogonal set \( \{ Q_w : w \in V' \} \) are constant on \( V \)-cosets), it is sufficient to show that every basis function \( Q_w(x), w \in V' \) (hence every linear combination of them) is constant on cosets of \( V \). Each \( V \)-coset is of the form \( y + V = \{ y \oplus z : z \in V \} \), \( y \in S \). On this coset, \( Q_w(x) = (-1)^{wy}z^t = (-1)^{wy}(y+z)^t = (-1)^{wy}z^t \) because \( w_1z^t = 0 \) for every \( z \) in \( V \) if \( w_1 \) is in \( V' \). Therefore, \( Q_w(x) \) has the same value \( (-1)^{wy}z^t \) on every point of \( y + V \). Furthermore, there are \( 2^n - k \) linearly independent functions \( Q_w(x) \), and these are a sufficient basis for the space of all functions on cosets of \( V \).

Example 9: Let \( n = 3 \) and let \( V \) be the subspace of \( \mathbb{Z}^3 \) consisting of all 3-tuples of even parity. That is, \( V = \{(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0)\} \). The cosets of \( V \) are \( V \) itself and the set \( V' = \{(0, 0, 0), (0, 1, 0), (1, 0, 0), (1, 1, 0)\} \). The subspace of functions that are constant on \( V \) and \( V' \) has a basis consisting of the function \( f(x) = 1 \) on \( V \) and the constant function \( f = 1 \).
Take the transform of $f$:

\[
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1 \\
1 & 1 & -1 & -1 \\
1 & -1 & -1 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -1 & 1 & -1 \\
1 & 1 & -1 & -1 \\
1 & -1 & -1 & 1
\end{bmatrix}
= \begin{bmatrix}
1 & 1 & +1 & +1 \\
1 & -1 & +1 & -1 \\
1 & 1 & -1 & -1 \\
1 & -1 & -1 & 1
\end{bmatrix}
= (4, 0, 0, 0, 0, 0, 4) \quad (40)
\]

In other words, the only nonzero transform coefficients are those for which $w = (0, 0, 0)$ and $w = (1, 1, 1)$. Each of these vectors satisfies $w x^t = 0$ for all $x$ in $V$, as stated in the theorem.

**COROLLARY 2.1.** Any function on $\mathbb{Z}^n$ whose nonzero transform coefficients occur for a subset $\{w : f^*(w) \neq 0\}$ of rank $n - k$ can be defined as a function of at most $(n - k)$ binary variables $(y_1, \ldots, y_{n-k})$ which are linear combinations of $x_1, \ldots, x_n$ (over $\mathbb{Z}$).

**Proof:** Let $V'$ be the subspace generated by $\{w : f^*(w) \neq 0\}$. Since a subspace $V$ is the nullspace of its nullspace $V'$, the subspace $V'$ must be the nullspace of some $k$-dimensional subspace $V$, and the preceding theorem implies that $f(x)$ is constant on cosets of $V$. That is, $f(x) = f(u)$ for any $x \in (u + V')$ and some unique $u \in S$ (the set of coset leaders for $V$). If $B$ is an $(n - k)$ by $n$ matrix whose rows are a basis for $V'$, then every $y \in V'$ is a linear combination $y B$ of $B$-rows for some $y \in \mathbb{Z}^{n-k}$. Now let $C = \begin{bmatrix} B \\ A \end{bmatrix}$ where $A$ is a basis for $V$.

Then every $x \in V$ can be expressed as $v = z A$ for some $z \in \mathbb{Z}^k$. Finally, $x = (y_1, \ldots, y_{n-k}, z_1, \ldots, z_k) \cdot C$ so that $(y; z) = x C^{-1}$. Let $D$ be the first $(n - k)$ columns of $C^{-1}$. Then $y = x D$ defines $2^{n-k}$ vectors $y \in \mathbb{Z}^{n-k}$ in 1:1 correspondence with elements of $V'$. The correspondence $g(y) = g(x D) = f(x)$ defines $g$ (and $f$) uniquely for any $f$ which is constant on cosets of $V'$; therefore, $f(x)$ has been expressed as a (degenerate) function of $n - k$ essential arguments.

**Example 10.** The function $f(x) = 1$ on $V$, 0 elsewhere of the preceding example has a transform with only two nonzero coefficients: $f^*(w) = 4$ for $w = (0, 0, 0)$ and $(1, 1, 1)$. This set has rank $n - k = 1$; therefore, by the
corollary, $f(x)$ is a function of one variable $y$. This is easily verified, because $x \in V$ iff $(xa) = 0$, where $a = (1, 1, 1)$. Therefore, $y = xa^t$ is a sufficient linear function, and $f(x) = \bar{y} = xa^t \oplus 1$.

**COROLLARY 2.2.** Let $V + c$ be any coset of any $k$-dimensional subspace $V$ of $\mathbb{Z}^n$, with $c \in V'$. Then the characteristic function $v_c(x)$ ($v_c(x) = 1$ on $V + c$ and 0 elsewhere) has the Fourier transform

$$v_c^*(w) = 2^k \delta_{w_0}(1)^{w_1}$$

(41)
i.e., $v_c^*$ is 0 except on $V'$, where it has magnitude $2^k$ and sign $(-1)^{w_1}$.

**Proof:** Following the theorem proof, let $w = w_1 \oplus w_2$, $x = y \oplus z$, with $w_1$ and $y \in V'$, $w_2$ and $z \in V$. Then $v_c(x) = \delta_{yc}$ and

$$v_c^*(w) = \prod_x (-1)^{w_1x} \prod_y (-1)^{w_2y} \delta_{yc} = 2^k \delta_{w_0}(1)^{w_1}$$

(42)

**COROLLARY 2.3.** The characteristic function $v(x)$ of a subspace $V$ of dimension $k$ has the Fourier transform $v^*(w) = 2^k$ on $V''$ and 0 elsewhere.

**Proof:** Let $c = 0$ in the preceding corollary.

**Example 11.** The function $f(x)$ in the preceding example is unity on the subspace $V = \{(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0)\}$. Therefore its transform should be

$$f^*(w) = v_c^*(w) = 2^2 \delta_{w_0}(1)^{w_1}$$

(43)

Here $c = 0$ and $w_1 = (1, 1, 1)$ and $(0, 0, 0)$ are the only two vectors such that $w_1x^t = 0$ for all $x$ in $V$. Therefore, $f^*(w) = 0$ for $w_2 \neq 0$ ($w \neq w_1$) and $f^*(w) = 4$ for $w = w_1$. This is the same result calculated directly in the example following the main theorem.

The preceding theorem and its corollaries are used iteratively in the algorithms of Section III. Therefore, its computational implication will be mentioned here. By the quotient group character theorem, if a function is constant on cosets of $V$, then it can be sampled once on each coset, and its transforms can be generated on $V'$. In most of our applications, both $V$ and $V'$ will have a basis of unit vectors, and vectors $y$ in $V''$ are in $1:1$ correspondence with $(n-k)$-tuples. In this case, an algorithm to generate a list of
$2^{n-k}$ pointers $i(j)$ such that $x(i(j)) \in V'$ can be constructed in approximately $2^{n-k}$ indexed add operations. A table lookup involving these $2^{n-k}$ components will permit $f_{i(j)}$ to be extracted from its $2^n$-dimensional array and packed into a $2^{n-k}$-dimensional array, after which its transform can be computed in $(n-k)2^{n-k+1}$ rather than $n2^{n+1}$ operations.

3. The Poisson Summation Theorem

Another important property of Fourier transforms, which plays only a minor role in classical applications but plays a major role in the theory of group characters, is the group-theoretic analog of the Poisson summation theorem (Loomis, 1953, p. 152). This theorem states that the average of a function's values over a subgroup $H$ of the domain $G$ may be computed by summing its Fourier transform over the quotient group $G/H$, with suitable normalization. It is far from obvious that the Poisson summation theorem for the real domain $\mathbb{R}$ is a specialization of this general group-theoretic statement.

There is a direct correspondence between subspaces of $\mathbb{Z}^n$ and additive subrings of $\mathbb{R}^n$. This correspondence provides additional insight on the analogous roles of harmonic analysis for functions of binary $n$-tuples and functions on $\mathbb{R}^n$. We will first state and interpret this theorem for the real domain $\mathbb{R}$.

If $f(x)$ is a continuous bounded function whose squared norm ($\int |f|^2 \, dx$) is finite on $\mathbb{R}$, then its Fourier transform $T_f(y)$ is defined for every $y$ in $\mathbb{R}$ by

$$T_f(y) = \int_{-\infty}^{\infty} f(x)\phi_y(x) \, dx$$

(44)

where $\phi_y(x) = \exp(2\pi i y x)$ is a character for every $y$ in $\mathbb{R}$. The Poisson summation formula states that, for any positive real number $a$ and a suitably restricted function $f(x)$,

$$\sum_{n \in I} f(na) = \frac{1}{a} \sum_{k \in I} T_f(k/a)$$

(45)

where $I$ is the ring of all integers.

One classical application of this theorem has been to convert a slowly convergent power series into a rapidly convergent one, by going to the transform domain. In our application, a sum over a rather large set of $2^n$ subspace elements $x \in V$ will be converted into a sum over the factor group $\mathbb{Z}^n/V$ whose size $2^{n-k}$ is inversely proportional to the size of $V$. To verify that this formula (for the real domain) actually makes a statement about subgroups and quotient groups, define $g(x) = \sum_{n \in I} f(x + na)$. Then $g(x)$ is periodic with period $a$, so $g(x)$ may be regarded as a function on the group
\( R/aR \) of real numbers under mod \( a \) addition. The set of numbers \( \{ka, k \in I\} \) is a subgroup \( H = aI \) of the group \( G = R \), since it is closed under addition, and the domain of \( g(x) \) is the quotient group \( G/H = R/aI \) of residue classes of real numbers under addition modulo \( a \). The character group of \( G = R \) is
\[
\chi_G = \{ \phi_y : \phi_y(x) = \exp(2\pi i y x), \quad y \in R \} \tag{46}
\]
The character group of \( G/H = R/aI \) is defined (without proof) as:
\[
\chi_{G/H} = \{ \psi_k : \psi_k(x) = \exp(2\pi i (k/a)x), \quad k \in I \} \tag{47}
\]
Since
\[
\psi_k(x) = \phi_{k/a}(x) \tag{48}
\]
\( \chi_{G/H} \) may be identified with the following subset of \( \chi_G \):
\[
\chi_{G/H} = \{ \phi_y \in \chi_G : y = k/a \quad \text{for some} \quad k \in I \} \tag{49}
\]
Note that \( \chi_{G/H} \) consists of exactly those characters in \( \chi_G \) which are constant on the subgroup \( H = aI \) of all integral multiples of \( a \) in \( R \). The next theorem is the equivalent statement for the finite Abelian group \( \mathbb{Z}^n \).

**THEOREM 2.6 (Poisson Summation).** If \( V \) is any subspace of \( \mathbb{Z}^n \), \( V' \) is the nullspace of \( V \), and \( f \) is a real- or complex-valued function on \( \mathbb{Z}^n \), then
\[
\sum_{x \in V} f(x) = 2^{k-n} \sum_{w \in V'} f^*(w) \tag{50}
\]
We shall first prove a generalization of this theorem, which defines the sum of \( f(x) \) over a coset \( V + c \) of \( V \). In Section III, this generalization will be derived from the convolution theorem. It provides a direct answer to the question “Which cosets of \( V \) are implicants of \( f \) or \( f^* \)?”.

**Proof:** Define \( g(x) = \sum_{x \in V} f(x \oplus z) \). Note that \( g \) is constant on cosets of \( V \) since, for any \( u \) in \( V \), \( \sum f(x \oplus z) = \sum f((x \oplus u) \oplus z) \) summed over \( V \). As in the previous section, let \( x = y \oplus z, w = w_1 \oplus w_2 \) denote the unique decomposition of \( x \) and \( w \) into elements of \( S \) or \( V' \) and \( V \), respectively. By the quotient group character theorem (Section II,B,2), the transform of \( g \) is defined by
\[
g^*(w) = 2^k \delta_{w_0} \sum_{y \in S} (-1)^{w_1} g(y) = 2^k \delta_{w_0} \sum_{y \in S} \sum_{z \in V} (-1)^{w_1} f(y \oplus z) = 2^k \sum_{x \in \mathbb{Z}^n} (-1)^{w_1} \delta_{w_0} f(x) \tag{51}
\]
using the identity which follows from the definition of \( y, z, w_1, \) and \( w_2 \):
\[
\delta_{w_0}(-1)^{w_1} = \delta_{w_2}(w_2) \quad \text{for} \quad x = y \oplus z, w = w_1 \oplus w_2, \text{and} \quad zw_1 = 0.
\]
Now multiply both sides by \((-1)^{w \cdot c}\) and sum over \(\mathbb{Z}^n\) with respect to \(w\) (\(c\) is a constant vector in \(\mathbb{Z}^n\)):

\[
\sum_w g^* (w) (-1)^{w \cdot c} = 2^k \sum_{w \in \mathbb{Z}^n} (-1)^{w \cdot c} \delta_{w_2 0} \sum_{x \in \mathbb{Z}^n} f(x) (-1)^{w \cdot c}
\]  
(52)

The left side is \((g^*)^* = 2^k g(c)\), and the right side is

\[
2^k \sum_{w \in \mathbb{Z}^n} \delta_{w_2 0} (-1)^{w \cdot c} f^* (w) = 2^k \sum_{w_1 \in \mathbb{Z}^n} (-1)^{w_1 \cdot c} f^* (w_1)
\]  
(53)

This proves the following generalization of the Poisson summation theorem:

\[
2^n \sum_{z \in \mathbb{Z}^n} f(z \oplus c) = 2^{k-n} \sum_{w_1 \in \mathbb{Z}^n} (-1)^{w_1 \cdot c} f^* (w_1)
\]  
(54)

The special case \(c = 0\) is the statement of the theorem.

4. Miscellaneous Properties

Other properties of Fourier transforms on finite Abelian groups are either more specialized or will play only a minor role in the applications herein. Among the latter are the Parseval identity \((f, g) = (f^*, g^*)\) which relates inner products of functions on \(\mathbb{Z}^n\) to inner products of their transforms. Another is Bessel's inequality, which states the monotone convergence of the mean squared error in approximating a function as a linear sum over a subset of the characters on \(G\), as the subset is expanded to include more and more basis functions. A third is the mean squared error minimization property, which states that in any approximation of a function as a linear combination of some subset of the character functions, the Fourier transform coefficients are optimal in the sense of yielding minimum mean squared error.

All of these properties are important on infinite-dimensional function spaces, but are trivial consequences of the orthogonality property of the characters as a basis for a finite-dimensional vector space. In particular, note that the mean squared error minimization property is not significant for \((0, 1)\)-valued functions, although it is for real-valued functions. When a binary function is approximated by a partial sum of its Fourier basis vectors, the error at any point is a multiple of \(2^{-n}\), and its sum of squares depends primarily on the number of points at which \(f(x)\) disagrees with its "approximation."

C. FUNDAMENTAL THEOREM ON INVARIANCE

The theorems presented in this section were first applied to combinational logic by Ninomiya (1958). Their present form was introduced by Lechner (1968). The invariant properties of Fourier transforms of \(2\)-valued functions under the restricted affine group (RAG) (to be defined in Section IV) clearly
show that harmonic analysis provides the best context within which to analyze the equivalence relations induced by RAG and its subgroups (see Section IV.A.3).

The transform \( f_N^* \) of the translated function \( f_N(x) = \frac{1}{2} - f(x) \) is the most convenient form from which to analyze prototype equivalence relations, because it is symmetric under functional complementation: \((f)_N = (\frac{1}{2} - f) = -f_N\). The transform \( f^* \) of \( f \) and the invariants of Golomb (1959) possess closely related properties but do not exhibit them in such a convenient manner. Calculation of \( f^* \) or \( f_N^* \) by the fast Fourier transform algorithm of Section II.A.4 is straightforward.

Prototype transformations (elements of the group RAG defined in Section IV) apply affine transformations \( y = xA \oplus b \) to the arguments of a function \( g(y) \) and add linear polynomials \((x \alpha d \oplus c)\) to its output to produce another function \( f(x) = g(xA \oplus b) \oplus x \alpha d \oplus c \) in the same prototype class. Both \( f_N^* \) and \( f^* \) have the same invariant properties under affine domain encodings \( y = xA \oplus b \). However, the effect of adding \((x \alpha d \oplus c)\) to the function \( \text{output} \) is different for both. The first theorem on invariance will consider affine domain encodings alone. In this section, we will use \( \oplus \) to represent mod 2 addition (the symmetric difference operation) on the range of a function, to avoid confusion with \text{real} addition of functions and their Fourier transform expansions. Of course, addition on the domain \( X = Z^* \) is \text{always} interpreted mod 2. However, the symbol \( + \) is used in the exponent of \(( -1)\) because real and mod 2 addition have the \text{same} effect.

**THEOREM 2.7 (Domain Encodings).** Let \( g(x) \) be a known function from \( X \) into \( Z \). If \( f(x) \) is defined, for all \( x \), by \( f(x) = g(xA \oplus c) \), then the Fourier transforms of \( f \) and \( g \) are related as follows:

\[
\begin{align*}
  f^*(w) &= (-1)^{wd} g^*(wA^{-1}); \\
  g^*(w) &= (-1)^{wd} f^*(wA^{1}) \\
  f_N^*(w) &= (-1)^{wd} g_N^*(wA^{-1}); \\
  g_N^*(w) &= (-1)^{wd} f_N^*(wA^{1})
\end{align*}
\]

(55) (56)

where \( A^{-1} \) is the transposed inverse of \( A \) and \( d = cA^{-1} \).

**Proof:** In the definition of \( f^* \), substitute \((y \oplus c)A^{-1}\) for \( x \), \( g(y) \) for \( f(x) \), and sum over \( y \) instead of \( x \).

\[
\begin{align*}
  f^*(w) &= \sum (-1)^{wX^*} f(x) \\
  f^*(w) &= \sum (-1)^{wA^{-1}(y+c)} g(y) \\
  &= (-1)^{wA^{-1}c} \sum_y (-1)^{wA^{-1}y} g(y) \\
  &= (-1)^{wd} g^*(wA^{-1})
\end{align*}
\]

(57)
Replacing \( w \) by \( vA^t \) gives the converse relation after rewriting \( w \) for \( v \) and transposing:

\[
\begin{align*}
    f^* (vA^t) &= (-1)^{w_d} g^* (v) \\
    g^* (w) &= (-1)^{w_d} f^* (wA^t)
\end{align*}
\tag{58}
\]

The corresponding relation between \( f_N^* \) transforms is determined by substituting in the above expressions the definition

\[
    f_N^* (w) = (\frac{1}{2} - f)^* = 2^{n-1} \delta_{w_0} - (-1)^{w_d} g^* (wA^{-t})
\]

where \( \delta_{w_0} = 1 \) if \( w = 0 \) and 0 otherwise. Since

\[
    2^{n-1} \delta_{w_0} = (-1)^{w_d} 2^{n-1} \delta_{w_0}
\]

for any \( d \),

\[
    f_N^* (w) = (-1)^{w_d} [2^{n-1} \delta_{w_0} - g^* (wA^{-t})] = (-1)^{w_d} g_N^* (wA^{-t})
\tag{60}
\]

from which the converse is obtained as before.

**Example 12.** Let \( g(x) = x_1 x_2 \lor x_3 \); that is, \( g^{-1}(1) = \{1, 3, 5, 6, 7\} \). Suppose \( f(x) = g(xA \oplus c) \). Then \( f(x) = 1 \) iff \( g(y) = 1 \) for \( y = xA \oplus c \). The following table computes \( y \) from \( x \) for the specific transformation

\[
    A = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad c = (1, 10) \tag{61}
\]

<table>
<thead>
<tr>
<th>( i )</th>
<th>( x(i) )</th>
<th>( xA )</th>
<th>( y(j) = xA \oplus c )</th>
<th>( f )</th>
<th>( g(y) = f(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0, 0, 0)</td>
<td>(0, 0, 0)</td>
<td>(1, 1, 0)</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>(0, 0, 1)</td>
<td>(0, 0, 1)</td>
<td>(1, 1, 1)</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>(0, 1, 0)</td>
<td>(0, 1, 1)</td>
<td>(1, 0, 1)</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>(0, 1, 1)</td>
<td>(0, 1, 0)</td>
<td>(1, 0, 0)</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>(1, 0, 0)</td>
<td>(1, 1, 0)</td>
<td>(0, 0, 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>(1, 0, 1)</td>
<td>(1, 1, 1)</td>
<td>(0, 0, 1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>(1, 1, 0)</td>
<td>(0, 1, 0)</td>
<td>(0, 1, 1)</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>(1, 1, 1)</td>
<td>(1, 0, 0)</td>
<td>(0, 1, 0)</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Thus, the function \( f(x) = 1 \) for \( i = 0, 1, 2, 5, 6 \).

Using the algorithm of Section II.4 we obtain \( g^* \) and \( f^* \):

\[
\begin{align*}
    g &= (0, 1, 0, 1, 0, 1, 1, 1) \\
    h &= g(Q_1 \times I^{[2]}) = (1, -1, 1, -1, 1, -1, 2, 0) \\
    r &= h(I^{[1]} \times Q_1 \times I^{[2]}) = (2, -2, 0, 0, 3, -1, -1, -1, -1) \\
    g^* &= r(I^{[2]} \times Q_1) = (5, -3, -1, -1, -1, -1, 1, 1) \\
    f &= (1, 1, 1, 0, 0, 1, 1, 0) \\
    h &= f(Q_1 \times I^{[2]}) = (2, 0, 1, 1, 1, -1, 1, 1) \\
    r &= h(I^{[1]} \times Q_1 \times I^{[3]}) = (3, 1, 1, -1, 2, 0, 0, -2) \\
    f^* &= r(I^{[2]} \times Q_1) = (5, 1, 1, -3, 1, 1, 1, 1)
\end{align*}
\tag{62}
\]
The theorem states that $g^*(w) = f^*(w \cdot A^t)(-1)^{w \cdot c}$. The image $wA^t$ and the sign change factor $(-1)^{w \cdot c}$ are calculated below for each $w$, with

$$A^t = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad c = (1, 1, 0)$$

<table>
<thead>
<tr>
<th>$i$</th>
<th>$w(i)$</th>
<th>$wA^t$</th>
<th>$j$</th>
<th>$wc^t$</th>
<th>$(-1)^{wc^t}$</th>
<th>$f^<em>(wA^t) = f_j^</em>$</th>
<th>$g^<em>(w) = g_1^</em>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0, 0, 0)</td>
<td>(0, 0, 0)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>(0, 0, 1)</td>
<td>(0, 1, 1)</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>$-3$</td>
<td>$-3$</td>
</tr>
<tr>
<td>2</td>
<td>(0, 1, 0)</td>
<td>(1, 1, 0)</td>
<td>6</td>
<td>1</td>
<td>$-1$</td>
<td>$+1$</td>
<td>$-1$</td>
</tr>
<tr>
<td>3</td>
<td>(0, 1, 1)</td>
<td>(1, 0, 1)</td>
<td>5</td>
<td>1</td>
<td>$-1$</td>
<td>$+1$</td>
<td>$-1$</td>
</tr>
<tr>
<td>4</td>
<td>(1, 0, 0)</td>
<td>(1, 0, 0)</td>
<td>4</td>
<td>1</td>
<td>$-1$</td>
<td>$+1$</td>
<td>$-1$</td>
</tr>
<tr>
<td>5</td>
<td>(1, 0, 1)</td>
<td>(1, 1, 1)</td>
<td>7</td>
<td>1</td>
<td>$-1$</td>
<td>$+1$</td>
<td>$-1$</td>
</tr>
<tr>
<td>6</td>
<td>(1, 1, 0)</td>
<td>(0, 1, 0)</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>$+1$</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>(1, 1, 1)</td>
<td>(0, 0, 1)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$+1$</td>
<td>1</td>
</tr>
</tbody>
</table>

Thus the theorem produces the same relation between $f^*$ and $g^*$ that was verified by direct calculation.

The above conversion relations show that linear transformations $x \rightarrow xA$ and vector addition $x \rightarrow x \oplus c$ affect $f^*$ (and $f_n^*$) in distinct and separate ways. When $A$ permutes $X$, its transposed inverse $A^{-t}$ permutes transform coordinates $w$. It is well-known that $A$ and $A^{-t}$ have the same cycle structure (Elspas, 1959). When $c$ is added to $x$, points of $X$ are permuted in pairs. However, the effect on $w$ is merely to change the signs of those transform coefficients $f^*(w)$ for which the dot product $wc^t$ has odd parity.

When $A$ is restricted to be a permutation matrix $P$, the results are noteworthy. Consider the subset of $n!/(n - k)!k!$ points $(x_1, \ldots, x_n) = x$ in $X$ that have weight $k$ (i.e., exactly $k$ of the $x_i$'s have unit value and the remaining $(n - k)$ variables are zero). This set is closed under variable permutations $x \rightarrow xP$, but not under variable complementations $(x \rightarrow x \oplus c)$ or symmetry transformations $(x \rightarrow xP \oplus c)$. On the other hand, the corresponding set of points of weight $k$ in $W$ remains closed under general symmetry transformations. The effect of a nonzero $c$ is merely to change the sign of $f^*$ or $f_n^*$ at certain points, not to permute the ordering of coefficients.

$$g^*(w) = (-1)^{w \cdot c} f^*(wP^t) \quad \text{if} \quad f(x) = g(xP \oplus c) \quad (63)$$

This property was used by Ninomiya and Golomb to detect the symmetry type to which a function belongs. It also provides necessary conditions for partial symmetry of a function with respect to a subset of variables or their complements and can even be applied to detect symmetry of $f$ with respect to linear combinations of variables (mod 2).
The second fundamental property of Fourier transforms is their behavior under additive linear functions on their range. This property is more complicated to represent in terms of \( f^* \) so we consider only \( f_N^* \). Suppose we desire to find \( (f \oplus g)_N^* \) in terms of \( f_N^* \), where \( g \) is the linear function \( g(x) = xa' \), and addition is mod 2. We first convert \( f \oplus g \) (mod 2) to the real expression

\[
f \oplus g \quad \text{(mod 2)} = f(1 - g) + g(1 - f) = f + g - 2fg \quad \text{(real arithmetic)} \tag{64}
\]

Therefore, \( 1 - 2(f \oplus g) = (1 - 2f)(1 - 2g) \) is an identity for all \( x \). We also note that the basis function \( Q_w(x) = (-1)^{wx'} \) can be represented as \( Q_w(x) = (1 - 2wx')^* = 2(wx')_N^* \). (Here the dot product \( wx' \) must be reduced mod 2 to 0 or 1.) The transform of \( f \oplus g \) after translation now becomes

\[
(f + g)_N^* = \left[ \frac{1}{2} - (f \oplus g) \right]^* = \left[ \frac{1}{2}(1 - 2f)(1 - 2g) \right]^* = \sum_x \frac{1}{2}(-1)^{wx'}(1 - 2f(x))(1 - 2g(x)) = \frac{1}{2} \sum_x (1 - 2f(x))(1 - 2xa')(1 - 2xw') \tag{65}
\]

Now apply the relationship \( 1 - 2(f \oplus g) = (1 - 2f)(1 - 2g) \) to the last two factors:

\[
(f \oplus g)_N^*(w) = \frac{1}{2} \sum_x (1 - 2f(x))(1 - 2(xa' \oplus xw')) = \frac{1}{2} \sum_x (1 - 2f(x))(-1)^{(w + a)x'} = f_N^*(w \oplus a) \tag{66}
\]

This, together with our previous observation that functional complementation merely reverses the sign of \( f_N^*(w) \) for all \( w \), proves the second theorem on invariance:

**THEOREM 2.8 (Range Encoding).** If \( f(x) \) is defined for all \( x \) as \( f(x) = g(x) \oplus xa' \oplus d \) (mod 2), then \( f_N^*(w) = (-1)^d g_N^*(w \oplus a) \). In other words, addition of linear functions \( (f \rightarrow f \oplus xa') \), which has the effect of changing the signs of certain coordinates of \( f_N \), permutes the transform coefficients in pairs (i.e., permutes the Fourier basis functions \( Q_w(x) \). This effect of linear functions is dual to the effect of vector addition \( (x \rightarrow x \oplus a) \) which permutes points of the function domain \( X \) (permutes the fundamental product or minterm basis for \( f(x) \)) but merely changes the signs of \( f_N^* \) coefficients.
**Example 13.** The function \( g(x) = x_1 \lor x_3 \) with \( g^{-1}(1) = \{1, 3, 5, 6, 7\} \) will be used again to illustrate range translation. Define \( f(x) = g(x) \oplus xa' \oplus d \) with \( a = (1, 0, 1) \) and \( d = 1 \). The effect of this mapping on \( g \) is computed below.

<table>
<thead>
<tr>
<th>( i )</th>
<th>( x(i) )</th>
<th>( g_i )</th>
<th>( xa' \oplus 1 = f_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0, 0, 0)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>(0, 0, 1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>(0, 1, 0)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>(0, 1, 1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>(1, 0, 0)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>(1, 0, 1)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>(1, 1, 0)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>(1, 1, 1)</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The transform of \( f \) is computed below:

\[
\begin{align*}
    f &= (1, 1, 1, 1, 0, 0, 1, 0) \\
    h &= f(Q_1 \times I^{[2]}) = (2, 0, 2, 0, 0, 0, 1, 1) \\
    r &= h(I^{[1]} \times Q_1 \times I^{[1]}) = (4, 0, 0, 0, 1, 1, -1, -1) \\
    f^* &= r(I^{[2]} \times Q_1) = (5, 1, -1, -1, 3, -1, 1, 1)
\end{align*}
\]

(67)

Using the range encoding theorem, \( f_N^*(w) = (-1)^d g_N^*(w \oplus d) \); the same result is obtained from \( g_N^* \) as follows (taking \( g^* \) from the preceding example):

<table>
<thead>
<tr>
<th>( i )</th>
<th>( w(i) )</th>
<th>( g_i^* )</th>
<th>( g_N^*(w) )</th>
<th>( w \oplus a )</th>
<th>(-g_N^<em>(w \oplus a) = f_N^</em>(w))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(0, 0, 0)</td>
<td>5</td>
<td>-1</td>
<td>(1, 0, 1)</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>(0, 0, 1)</td>
<td>-3</td>
<td>3</td>
<td>(1, 0, 0)</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>(0, 1, 0)</td>
<td>-1</td>
<td>1</td>
<td>(1, 1, 1)</td>
<td>+1</td>
</tr>
<tr>
<td>3</td>
<td>(0, 1, 1)</td>
<td>-1</td>
<td>1</td>
<td>(1, 1, 0)</td>
<td>+1</td>
</tr>
<tr>
<td>4</td>
<td>(1, 0, 0)</td>
<td>-1</td>
<td>1</td>
<td>(0, 0, 1)</td>
<td>-3</td>
</tr>
<tr>
<td>5</td>
<td>(1, 0, 1)</td>
<td>-1</td>
<td>1</td>
<td>(0, 0, 0)</td>
<td>+1</td>
</tr>
<tr>
<td>6</td>
<td>(1, 1, 0)</td>
<td>1</td>
<td>-1</td>
<td>(0, 1, 1)</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>(1, 1, 1)</td>
<td>1</td>
<td>-1</td>
<td>(0, 1, 0)</td>
<td>-1</td>
</tr>
</tbody>
</table>

From \( f_N^* \) we obtain \( f^* = (5, +1, -1, -1, +3, -1, +1, +1) \) which agrees with the preceding direct computation.

We have discussed the separate effects of domain and range transformations which comprise the elements of RAG. From these two results, the following fundamental theorem is easily derived:
THEOREM 2.9 (Fundamental Invariance Theorem). Let $g(x)$ be a known function from $X = \mathbb{Z}^n$ into $Z = \{0, 1\}$. Suppose $f(x)$ is defined for all $x$ by

$$f(x) = g(xA \oplus c) \oplus xa^t \oplus d \pmod{2}$$

where $A$ is a nonsingular $n$ by $n$ matrix over $GF(2)$, $a$ and $c$ are arbitrary binary $n$-tuples, and $d \in Z$. Then the transforms of $f_N$ and $g_N$ are related as follows:

$$N(x) = (-1)^{xa^t \oplus d} g_N(xA \oplus c)$$

iff

$$g_N^*(w) = (-1)^{wc^t \oplus d} f_N^*(wA^t \oplus a)$$

(68)

Proof: Let $h(x) = g(xA \oplus c)$ and $f(x) = h(x) \oplus xa^t \oplus d$. Then by the second invariant property of $f_N^*$ (under range transformations)

$$f_N^*(w) = (-1)^d h_N^*(w \oplus a)$$

(69)

By the first invariant property of $f_N^*$ (under domain transformations):

$$h_N^*(w) = (-1)^{wb^t} g_N^*(wA^{-t})$$

(70)

where $b = cA^{-1}$. Combining the two expressions, we obtain

$$f_N^*(w) = (-1)^{d \oplus (w \oplus d) b^t} g_N^*((w \oplus a)A^{-t})$$

(71)

Substituting $v = (w \oplus a)A^{-t}$, we obtain

$$f_N^*(wA^t \oplus a) = (-1)^{d \oplus vA^t} g_N^*(v)$$

(72)

which becomes the inverse relation upon replacing $v$ by $w$ and $b$ by $cA^{-1}$:

$$g_N^*(w) = (-1)^{wc^t \oplus d} f_N^*(wA^t \oplus a)$$

(73)

To derive the symmetric companion statement relating $f_N$ to $g_N$, apply the identity $1 - 2k = (-1)^k$ for $k = 0, 1$, to the function $k(x) = xa^t \oplus d \pmod{2}$. Then $f = h \oplus k = h + k - 2kh$ (real addition), so the theorem hypothesis becomes

$$f_N(x) = \frac{1}{2} - f(x) = (1 - 2k(x))(\frac{1}{2} - h(x))$$

$$= (-1)^{xa^t \oplus d} [\frac{1}{2} - g(xA \oplus c)] = (-1)^{xa^t \oplus d} g_N(xA \oplus c).$$

(74)

The symmetry of the theorem statement makes it easy to remember:

$$f_N(x) = (-1)^{xa^t \oplus d} g_N(xA \oplus c) \quad \text{iff} \quad g_N^*(w) = (-1)^{wc^t \oplus d} f_N^*(wA^t \oplus a)$$

(75)

The second expression is deducible from the first one simply by interchanging $f_N$ with $g_N$ and $a$ with $c$, then replacing $f_N$, $g_N$, and $A$ by $f_N^*$, $g_N^*$, and...
A^t$, respectively. We have thus demonstrated a complete two-way duality between (a) adding a constant vector $b$ to every point of $X$ and (b) adding a linear function $xa^t$ to every function in $F$. This duality is obvious only when the range of $F$ is translated from $\{0, 1\}$ to the symmetric set $\{\frac{1}{2}, -\frac{1}{2}\}$, in which case mod 2 addition of functions $f \oplus g$ becomes multiplication $(f \oplus g)_N = 2f_N g_N$. This is another justification for preferring the modified transform $f_N^* = (\frac{1}{2} - f)^*$ rather than the unmodified Fourier transform $f^*$ to represent a 2-valued function $f$.

To recapitulate, the fundamental invariance theorem says that when an element $T_{A,a,c,d}$ of RAG is applied to a function $f$, it has the following effects on the transform $f_N^*$: Parameters $A$ and $a$ permute the components of the transform coefficient vector $f_N^*$, while parameters $c$ and $d$ merely change their signs. Furthermore, $A$ does not move the 0-coefficient $f_N^*(0)$, while $a$ interchanges $f_N^*(0)$ and $f_N^*(a)$. In Section V, these properties will be used to guide the selection of a sufficient encoding transformation to map any function into its prototype, or vice versa.

**EXERCISES**

1. Prove the recursion relation $Q_{n+1} = Q_1 \times Q_n$ by relating the radix-two expansions of the indices $i$ and $j$ of $q_{ij}$ to the argument vectors $x, w$ of $Q_n(x)$.

2. Generate $Q_3$ and take the Fourier transforms of the functions $f, g, h$ defined on $\mathbb{Z}^3$ by the modulo 2 sums $x_1 \oplus x_2, x_1 \oplus \bar{x}_3, \bar{x}_2 \oplus \bar{x}_3$. What can be said about the distribution of $f^*$ coefficient magnitudes for these three functions?

3. Show that $\sum \{(-1)^{x} : v \in \mathbb{Z}^n\} = 0$ when summed over all $v$ in $\mathbb{Z}^n$.

4. Give a direct proof that $\sum \{(-1)^{v_b} : v \in \mathbb{Z}^n\} = 2^n \delta_{b0}$. Hint: Express $v$ as $v_1 \oplus v_2$ with $v_1 \leq b$ and $v_2 \leq \bar{b}$.

5. Define the matrix $Q_4$ and compute the Fourier transform $f^* = FQ_4$ of the 4-argument function $f(x)$ with $f^{-1}(1) = \{0, 2, 3, 5, 6, 8, 9, 11, 12\}$ by direct multiplication. Verify the matrix identity $F^* = Q_2 FQ_2$, where $F$ and $F^*$ are $4 \times 4$ matrices in which the elements of $f$ and $f^*$, respectively, are inserted row by row (Hint: See Fig. 4).

6. Compute the Fourier transform of the function $g^{-1}(1) = \{7, 9, 12, 13, 15\}$ using the $4 \times 4$ matrix equation $G^* = Q_2 GQ_2$.

7. Find the convolution sum $f^* \cdot g$ of the functions $f$ and $g$ of the two preceding problems by means of the formula $(f^* \cdot g) = 2^{-4} Q_2 H^* Q_2$, where $H^*$ is the componentwise product of the matrices $F^*$ and $G^*$.
III. COMBINATORIAL APPLICATIONS

This section applies the Fourier transform properties described in Section II,B to three problems: prime implicant extraction, detection of disjunctive decompositions, and partial ordering of variables before factoring a logic function. These problems are called combinatorial because the classical approach to their solutions has generally been combinatorial in nature and because the cost of solving the problem (computation time, storage, or both) grows exponentially or factorially with the number of function arguments. Harmonic analysis has not yet been applied to the most significant problem of this type, the "minimal covering" problem (find the "least-cost" subset of prime implicants whose union includes all of \( f^{-1}(1) \) or \( f^{-1}(0) \)). However, there is a remarkable consistency among the transform techniques required to detect prime implicants, detect disjunctive decompositions, or select variables to be factored out. This unifying tendency alone warrants further exploitation of harmonic analysis as a source of new algorithms.

The algorithms developed in this section do not use any of the invariant properties of Fourier transforms under affine operators developed in Section II,C. Section IV will examine these properties, and Section V will describe a synthesis technique based on them. Historically, that technique was developed before the techniques in this section were discovered. However, this section is recommended reading before Section V because it provides a rationale for the criteria used in that section to select an equivalent function of minimal complexity.

Sections III,A,1 and III,A,2 develop some elementary concepts and notations which may be skipped by the reader whose background includes some switching theory or modern algebra.

A. A TEST FOR IMPlicants OF \( f \) OR \( \bar{f} \)

This section and the next one will develop a new algorithm that can simultaneously detect all implicants of a fully defined function \( f \) and its complement \( \bar{f} \). The algorithm is independent of the size of \( f^{-1}(1) \). If \( f \) has "don't-care" conditions, most of the computation involved to find implicants of \( f \) must be duplicated to find the implicants of \( \bar{f} \). In contrast to conventional methods, implicants are detected in decreasing rather than increasing order of their dimension. As each new implicant of dimension \( k \) is detected, it is compared to adjacent implicants (cosets of the same subspace), and redundant ones are discarded. Data is also kept from which the number of prime implicants or "PI's" which actually cover each minterm of \( f \) or \( \bar{f} \) can be evaluated, without
actually adding unity to each location in a truth table array covered by each PI.

A byproduct of the PI detection algorithm is a necessary condition for disjunctive decomposition of a function with respect to two complementary subsets of variables. This test will be described in Section III.A.3.

Section III.A.1 introduces standard notation. The reader who is unfamiliar with the terminology should refer to any standard text on switching theory such as Chapter 3 of Miller (1965) or Chapter 2 of Prather (1967). Section III.A.2 will be familiar to anyone with background in vector spaces over finite fields.

1. Introduction and Notation

The notation of Section II.A.1 will be used herein. A subcube of \( \mathbb{Z}^n \) is a subset with the following special properties:

1. Certain "bound" argument variables are restricted to a fixed value which may be 0 or 1.
2. The remaining "free" variables can independently take on all combinations of 0 or 1 values.

(The term "subcube" arises from the geometric picture of \( \mathbb{Z}^n \) as an \( n \)-dimensional unit cube (\( n \)-cube) having the \( 2^n \) binary \( n \)-tuples which represent elements of \( \mathbb{Z}^n \) as vertex coordinates.) The dimension of a subcube is the number of free variables it contains. For convenience, we use the term \( k \)-cell (or \( k \)-cube) to denote a \( k \)-dimensional subcube of \( \mathbb{Z}^n \). A subcube of dimension \( k \) includes \( 2^k \) points of \( \mathbb{Z}^n \).

A unique \( k \)-cell is denoted by the ternary \( n \)-tuple \( a = (a_1 \cdots a_n) \). The symbol "\( \cdot \)" is assigned to the \( k \) components \( a_i \) for which \( x_i \) is a free variable in \( \mathbb{Z}^n \); \( a_i \) is assigned the same 0 or 1 value as \( x_i \) for each of the \( n - k \) bound variables \( x_i \). Since every coordinate of \( \mathbb{Z}^n \) can be either free, fixed at 1, or fixed at 0, and every coordinate of \( a \) can independently assume one of three possible values (0, 1, or \( \cdot \)), there are exactly \( 3^n \) ternary \( n \)-tuples corresponding to \( 3^n \) possible subcubes. The total number of \( k \)-cells in \( \mathbb{Z}^n \) is \( 2^{n-k} C_{n,k} \), where \( C_{n,k} = n! / k! (n-k)! \) is the number of ways to select \( k \) free variables among the \( n \) coordinates of \( \mathbb{Z}^n \), and \( 2^{n-k} \) is the number of ways to assign values to the \( n-k \) bound variables.

An implicat of a logical function \( f \) (or of its complement \( \overline{f} \)) defined on \( \mathbb{Z}^n \) is a subcube of \( \mathbb{Z}^n \) which is also a subset of the level set \( f^{-1} (e) = \{ x \in \mathbb{Z}^n : f(x) = e \} \) for \( e = 0 \) or 1. A prime implicat of a function \( f \) is an implicat which cannot be doubled in size by freeing any one of its bound variables. An

\footnote{The symbol "\( \cdot \)" corresponds to "\( x \)" of Miller (1965) and "\( I \)" of Prather (1967).}
important step in the simplification of logic equations is to generate all prime implicants of \( f \) while eliminating all redundant nonprime ones.

The list of all prime implicants for \( f \) contains the information necessary to derive a disjunctive normal formula (DNF) of minimal complexity, i.e., the simplest Boolean sum of products expression for \( f \). Each prime implicant corresponds to an AND gate from which no literal can be omitted, because it will no longer correspond to a subset of \( f^{-1}(1) \). Although no prime implicant contains redundant literals, some prime implicants may be redundant in the covering of \( f \) by a union of prime implicants.

The second step in simplifying logic formulas is to find a subset of the prime implicants which is minimal in some sense but which still covers all of \( f \) (or \( f \)). In general, such a "minimal cover" is not unique. The problem of selecting a minimal cover from among the prime implicants of \( f \) will not be discussed here. However, the invariant properties of \( f^* \) under symmetries of the \( n \)-cube (Section II,C) suggests that harmonic analysis may have potential applications to the recognition of cyclic covers, which lead to difficulties in existing combinatorial approaches.

2. Cosets and Nullspaces

The algorithm to be described herein takes advantage of Fourier transform properties to separate the test "Is \( V_b + c \) included in \( f^{-1}(1) \)?" into two parts. The first part operates on a subset of \( f^* \) and produces an array of integers corresponding to the cosets of \( V_b \); the second part tests the integer corresponding to \( c \), and on this basis decides whether \( V_b + c \) implies \( f \) or \( f \).

The concept of a \( k \)-cell is inadequate to describe this algorithm, because it does not explicitly distinguish subspaces of \( Z^n \) from cosets of these subspaces. Subspaces (additive subgroups) of \( Z^n \) are merely implicants or \( k \)-cells with the added constraint that all bound variables are assigned zero value. This constraint insures that the subset so defined contains the zero vector (additive identity element) and is closed under addition. For example, \( a = (-000-) \) defines a subspace of \( Z^n \), while \( a = (-101-) \) does not.

**DEFINITION.** Let \( a \) denote a \( k \)-cell in which all bound variables are assigned zero value. The \( k \)-dimensional subspace defined by \( a \) will be represented by \( V_b \), where \( b \) is the binary \( n \)-tuple obtained from \( a \) by replacing the symbol - by 1. For example, if \( a = (-000-) \), then \( b = (10001) \), and \( V_b \) is the following subset of \( Z^5 \): \((00000),(00001),(10000),(10001)\). Note that the unit vectors \((10000),(00001)\) whose sum is \( b \) are a basis for \( V_b \).

Each subspace of \( Z^n \) defines a disjoint partition of \( Z^n \) into disjoint cosets. The coset \( V_b + c \) is obtained by adding some vector \( c \) in \( Z^n \) (but not in \( V_b \) to
every vector in $V_b$. This is called translation of $V_b$ by $c$. (The notation $S + c$ denotes the set \{x \oplus c : x \in S\}, for any subset $S$ of $\mathbb{Z}^n$.) For example, if $b = (10001)$ and $c = (01010)$, the coset $V_b + c = \{x \oplus c : x \in V_b\}$ is identical to the 2-cell $a = (-101-)$. Furthermore, if $V_b$ is a $k$-cell of $\mathbb{Z}^n$, then every one of the $2^{n-k}-1$ nonzero assignments of values for the bound variables of $V_b$ generates a different $k$-cell or coset of $V_b$, which together with $V_b$ itself exhaust $\mathbb{Z}^n$. It is standard practice to call $V_b$ the "identity" coset generated by the remaining (zero) assignment to all bound variables.

Since $V_b$ is closed under addition, $V_b + (x \oplus c) = (V_b + x) \oplus c = V_b + c$ for any $x$ in $V_b$. Therefore, $2^k$ different vectors $x \oplus c$ will translate $V_b$ into the same coset $(V_b + c)$. However, only one of these vectors has zero values for each of the free variables in $V_b$; from now on we will use the symbol $c$ to denote this (unique) translation vector.

The unique correspondence between a $k$-cell $a$ and a coset $V_b + c$ defined above is specified by the following table of correspondences between components of $a$, $b$, and $c$:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that components of $c$ corresponding to bound components of $a$ must agree in value with $a$. Components of $c$ which are free variables in $a$ are assigned the value zero.

The weight of a vector $x$ in $\mathbb{Z}^n$, denoted $|x|$, is defined as the number of coordinates with value 1, or the real sum $x_1 + \cdots + x_n$. Every vector except $c$ itself in the coset $V_b + c$ (with $c$ uniquely defined as above) is the translate of a nonzero element $x$ of $V_b$; therefore, it has unit value for at least one of the free variables in $V_b$. In other words, $|x \oplus c| > |c|$ for every nonzero $x$ in $V_b$.

Because $c$ is the unique vector of minimum weight in the coset $V_b + c$, it is called the coset leader (Peterson, 1961).

An alternate definition for $V_b$ is the set \{x : x \leq b\} of all elements of $\mathbb{Z}^n$ which satisfy the following partial ordering relation:

**DEFINITION.** For fixed $b \in \mathbb{Z}^n$, the notation $x \leq b$ means $x_i \leq b_i$ for $1 \leq i \leq n$.

Clearly $V_b = \{x : x \leq b\}$. If $|b| = k$, then $V_b$ has $k$ unit basis vectors and $2^k$ elements.
DEFINITION. The characteristic function of $V_b$, denoted $v_b(x)$, is defined by $v_b(x) = 1$ for $x$ in $V_b$ and 0 otherwise.

Now consider the set $V_b = \{ y : y \leq b \}$. This set is also a subspace, the sum of whose unit basis vectors is $b$, the logical complement of $b$. The unit bases for $V_b$ and $V_b$ generate all of $Z^n$, and $xy^t = 0$ for any $x$ in $V_b$ and $y$ in $V_b$. Therefore, every $x$ in $Z^n$ has a unique decomposition $x = x_1 \oplus x_2$, $x_1 \in V_b$, $x_2 \in V_b$, and $Z^n$ is the direct sum of $V_b$ and $V_b$. In this notation, an alternate definition for the characteristic function of $V_b$ is $v_b(x) = \delta_{x \leq 0}$. The components $x_1$, $x_2$ are called the projections of $x$ into $V_b$ and $V_b$, respectively (also denoted $x \mapsto V_b$, $x \mapsto V_b$).

DEFINITION. For any subspace $V$ of $Z^n$, the set $\{ y : y x^t = 0 \}$ for all $x \in V$ is called the nullspace of $V$.

Clearly, $V_b$ is the nullspace of $V_b$ (and vice versa). Furthermore, $V_b$ contains every coset leader for the partition of $Z^n$ into cosets of $V_b : Z^n = \{ V_b + c : c \in V_b \}$. Caution: unless $V_b$ has a basis of unit vectors, $V_b$ cannot always be identified with the set of coset leaders; and $Z^n$ is not necessarily the direct sum of $V_b$ and $V_b$. For example, the subspace $(00, 11)$ of $Z^2$ is its own nullspace (mod 2). Its only coset is $(01, 10)$.

The test for implicants will make use of another notational device: the restriction of a function $f$ to a subset $S$ of its domain.

DEFINITION. For any function $f$ on $Z^n$, and any subset $S \subseteq Z^n$, the restriction of $f$ to $S$ (denoted $f \upharpoonright S$) is defined as the set of ordered pairs

$$\{(y, f(y)) : y \in S\}.$$

If $S$ is a subspace of the type $V_b$ in $Z^n$, it will be convenient to specialize this notation considerably, using the projection $x \mapsto V_b$ defined previously. Suppose $|b| = k$. If we regard $x$ as a function from the integer index set $(1, 2, \ldots, n)$ into $Z$, then $x \mapsto V_b$ picks out the free variables $x_{r_j}$, $1 \leq j \leq k$ which correspond to unit components of $b$ $(b_{r_1}, b_{r_2}, \ldots, b_{r_k} = 1)$. The projection $x \mapsto V_b$ defines a new compact set of argument vectors in $Z^k$ for the function $f \mapsto V_b$; $V_b = (x_{r_1}, x_{r_2}, \ldots, x_{r_k}) = (y_1, \ldots, y_k)$. Then $f \mapsto V_b$ can be defined as the set of ordered pairs $\{(y, g(y)) : y = x \mapsto V_b, g(y) = f(x_i)\}$, where $x = x_1 \oplus x_2$, $x_1 \in V_b$, $x_2 \in V_b$, and $y$ merely compresses the vectors $x_i$ into $k$-tuples by deleting components of $x$ which are identically zero.
3. Main Theorem on Implicant Extraction

The main theorem on implicant extraction will now be stated and proved using the theorems of Section II.B.

**THEOREM 2.9 (Implicant Extraction).** For any subspace $V_b = \{x : x \leq b\}$ of $\mathbb{Z}^n$, and any coset leader $c$ in the nullspace $V_b$,

$$
V_b + c \subseteq f^{-1}(1) \quad \text{iff} \quad (f^* \cap V_b)_c^* = 2^n \quad (76)
$$

$$
V_b + c \subseteq f^{-1}(0) \quad \text{iff} \quad (f^* \cap V_b)_c^* = 0
$$

*Proof:* Let $k = |b|$, the dimension of $V_b$. For $z = 0$ or $1$, the condition $V_b + c \subseteq f^{-1}(z)$ is equivalent to

$$
\sum \{f(x) : x \in V_b + c\} = z2^k \quad (77)
$$

which may be rewritten in terms of the characteristic function $v_b$:

$$
\sum \{f(x) \oplus c) ; \quad x \in V_b\} = \sum \{f(x \oplus c)v_b(x) : x \in \mathbb{Z}^n\} = (f * v_b)_c \quad (78)
$$

by the definition of a convolution sum in Section II,B,1. By the convolution theorem (Section II,B,1)

$$
(f * v_b)_c = 2^{-n}(f*(w)v_b^*(w))_c \quad (79)
$$

The transform of $v_b(x)$ is $2^k$ on $V_b$ and 0 elsewhere by Corollary 2.3 of Section II,B,2. The remainder of the proof is clarified by adopting the following notation for the (unique) decompositions of $x$ and $w$ into direct sums of their projections into $V_b$ and $V_b$:

$$
x = c \oplus u, \quad c \in V_b, \quad u \in V_b \quad (80)
$$

$$
w = w_1 \oplus w_2, \quad w_1 \in V_b, \quad w_2 \in V_b \quad (81)
$$

Then

$$
(f * v_b)_c = 2^{-n}((2^k\delta_{w_20})(f*(w_1)))_c \quad (82)
$$

The inverse transform can also be decomposed into separate transforms on $V_b$ and $V_b$:

$$
(f * v_b)_c = 2^n \sum_{w_2} 2^k\delta_{w_20}(-1)^{w_20} \sum_{w_1} f^*(w_1)(-1)^{w_1c} \quad (83)
$$

$$
= 2^{-n}2^k \sum_{w_1} f^*(w_1)Q_{w_1}(c)
$$

$$
= 2^{k-n}(f^* \cap V_b)_c^*$$
This result also follows from the Quotient Group Character theorem of Section II,B,2, but the derivation above avoids the notational problems involved in an unambiguous application of this theorem. It has been shown that for any \( b \in \mathbb{Z}^n \), \( \sum \{ f(x) : x \in V_b + c \} = 2^{k} \cdot (f^* \Gamma V_c)_b^* \) for each \( c \) in \( V_b \). This sum will be \( 2^{k} \) or 0 if \( (f^* \Gamma V_c)_b^* = 2^k \) or 0, respectively, which proves the theorem.

This theorem permits the question "Is the coset (subcube) \( V_b + c \) an implicant of \( f \) or \( f^* \) ?" to be answered simultaneously for every coset of the subspace \( V_b \) merely by inspecting the results of a single inverse transform operation on the array \( (f^* \Gamma V_b) \), whose size is exactly equal to the number of cosets of \( V_b \).

4. Example of Theorem Operation

The example shown in Fig. 4 should clarify this theorem and remove any mystery surrounding its operation. Figure 4 shows the truth table of a function \( f(x) \) of five arguments arranged in a \( 4 \times 8 \) array \( F \), such that the first column of \( F \) corresponds to the subspace \( V_b, b = (11000) \). This subspace has eight 4-element cosets. The top row of \( F \) corresponds to the subspace \( V_b \). Each column of \( F \) is the restriction \( (f \Gamma V_b + c) \) of \( f \) to a different coset of \( V_b \).

\[
\begin{array}{cccccccc}
\text{F} & = & f(x) \\
\begin{array}{ccc}
(x_1, x_2, x_3, x_4) & \in & V_b \\
\text{c} & \in & V_b
\end{array}
\end{array}
\]

\[
\begin{array}{cccccccc}
00 & 01 & 01 & 10 & 10 & 01 & 11 & 11 \\
01 & 00 & 11 & 10 & 11 & 01 & 00 & 00 \\
10 & 11 & 11 & 11 & 11 & 11 & 11 & 11 \\
11 & 10 & 11 & 00 & 10 & 00 & 11 & 00 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\text{F} & = & f = f^*(w) \\
\begin{array}{ccc}
(x_1, x_2, x_3, x_4) & \in & V_b \\
\text{c} & \in & V_b
\end{array}
\end{array}
\]

\[
\begin{array}{cccccccc}
00 & 01 & 01 & 10 & 10 & 01 & 11 & 11 \\
01 & 00 & 11 & 10 & 11 & 01 & 00 & 00 \\
10 & 11 & 11 & 11 & 11 & 11 & 11 & 11 \\
11 & 10 & 11 & 00 & 10 & 00 & 11 & 00 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
20 & 04 & 00 & 06 & -2 & -2 & 6 & \\
04 & 00 & 06 & -2 & 04 & 00 & 06 & 04 \\
-2 & -2 & 22 & 00 & 00 & 00 & 00 & 00 \\
22 & 00 & 00 & 00 & 22 & 00 & 00 & 00 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\text{THEOREM INTERPRETATION:} \\
\sum (f \Gamma V_b + c) \subseteq (f \Gamma V_b + c) \subseteq (f \Gamma V_b + c) \subseteq (f \Gamma V_b + c) \\
\end{array}
\]

\[
\begin{array}{cccccccc}
\text{FIG. 4. Comparison of direct and transform approaches to implicant extraction for the subspace } V_{11000}. \\
\end{array}
\]
On the left side of Fig. 4, the sum of \( f \) over each of these columns is computed directly. Notice that for \( c = 0 \) and \( c = (00011) \), \( V_b + c \) is an implicant of \( f \), while for \( c = (00111) \), \( V_b + c \) is an implicant of \( f \) (since these are constant columns of \( F \)).

The array \( F^* \) on the right side of Fig. 4 is the Fourier transform \( f^* \) of \( f \). The Kronecker product definition of the fast Fourier transform algorithm in Section II, A, 4 can also be written in the following matrix form, in which \( Q^{[2]} \) transforms \( f \) on \( V_b \) and its cosets, \( Q^{[3]} \) transforms \( f \) on \( V_b \) and its cosets, and the multiplication is associative:

\[
F^* = Q^{[2]}FQ^{[3]} \tag{84}
\]

The restriction \( (f^* \cap V_b) \) is merely the top row of \( F^* \), and its inverse transform is merely the top row of

\[
2^{-3}F^*Q^{[3]} = 2^{-3}Q^{[2]}FQ^{[3]}Q^{[3]} = Q^{[2]}F \tag{85}
\]

since \( Q^{[3]}Q^{[3]} = 2^4I \).

The direct approach (sum over cosets of \( V_b \)) is also equivalent to the top row of \( Q^{[2]}F \), because the first row of \( Q^{[2]} \) is merely the vector \((1111)\).

In other words, identical results are obtained by both approaches. However, the transform algorithm required the definition of \( 2^k \) pointers followed by \( k2^k+1 \) additions to compute the inverse transform, while the direct approach requires \( 2^n \) pointers to be defined followed by \( 2^n \) additions to sum \( f \) over each coset.

A cumulative comparison of the cost \( 3k2^k \) for each subspace by the theorem versus the cost \( 2^{2^n+1} \) for each subspace by direct summation is also shown on Fig. 4. For \( n = 5 \), the two approaches are comparable. However, for \( n \geq 10 \), significant storage and computational advantages of the transform approach will be demonstrated in Sections III,B,5 and III,B,6.

B. ALGORITHM TO EXTRACT PRIME IMPLICANTS

This section describes an algorithm for extracting all prime implicants of \( f \) and \( \bar{f} \) based on the preceding theorem and computes an accurate bound on its computation requirements. As a starting point for this algorithm, the Fourier transform \( f^* \) of \( f \) is computed and stored either in fast (core) memory or on a disc file. The array \( f^* \) has \( 2^n \) integer elements in the range \((-2^{n-1}, +2^n)\), and it is the starting point for all further calculations.

The basic iteration step is applied to subspaces of the type \( V_b \) of \( Z^n \). Subspaces can be processed one at a time in any order. However, it is preferable to process subspaces in descending order of their dimension which corresponds to ascending order of the number of bound variables in their cosets. Therefore, subspace indices \( b \) will be processed in decreasing order of their weight \( |b| = n - k \). In other words, prime implicants containing \( k = 1 \) literals
will be extracted first, those with \(k = 2\) literals next, and so on. Prime impliants with \(n\) literals (minterms) will be extracted last.

This order of processing permits the algorithm to terminate after any value of \(k\) if it is found that \(f\) or \(\bar{f}\) or both are covered completely by prime impliants containing \(k\) or fewer literals.

Another reason for this order of processing is that it permits reversion to a more direct approach if desired as \(k\) approaches \(n\). The computational burden per subspace \(V_b\) increases as \((k + 1)2^k\); Section III,B,5 shows that a direct search approach involves less computation for the highest \((n/4)\) values of \(k\). The labor of the direct approach can be further reduced by identifying all minterms previously covered as "don't care" argument states. The direct approach then proceeds by inspecting the \(n\) "nearest neighbors" of each remaining (still uncovered) minterm.

Functions with "don't care" conditions will be considered in the last paragraph of Section III,B.

1. **Main Algorithm Definition**

The basic iteration process for each \(V_b\) with \(|b| = n - k\) consists of the following six steps:

1. A list of pointers \(\{ij: x(i) \in V_b \text{ and } x(i) \cap V_b = y(j), \ 0 \leq j < 2^k\}\) is constructed (e.g., using the algorithm of Section III,B,2 below) and saved for multiple uses.

2. The subarray \(f^* \cap V_b = \{f_j: 0 \leq j < 2^k\}\) is extracted from \(f^*\) by a table lookup process.

3. The inverse transform \((f^* \cap V_b)^*\) is computed by the fast Fourier transform algorithm of Section II,A,4.

4. The array \((f^* \cap V_b)^*\) is inspected at each argument \(y\) to detect impliants \(V_b + c\), as proscribed in the preceding theorem.

5. Elements of \((f^* \cap V_b)^*\) corresponding to the \(k\) "nearest neighbor" cosets of \(V_b + c\) are inspected to reject all redundant or nonprime impliants. All prime impliants are added to a list (which need not be retained in core memory).

6. (Optional) The Fourier transforms of prime impliants corresponding to all cosets of \(V_b\) are accumulated in an auxiliary array of \(2^n\) locations. This array defines the Fourier transform of the (real) sum of characteristic functions of prime impliants of \(f\) minus those of \(\bar{f}\). By inverting this transform, the number of prime impliants (so far detected) which cover each minterm of \(f\) and \(\bar{f}\) may be determined.

The next few paragraphs describe algorithms for steps (1), (5), and (6) in order to develop bounds on computation time for this procedure.
2. Generation of Subspace Pointers

In order to extract $f^* \cap V_b$ in step (1) and to update $g^*$ from $\Delta g^* \cap V_b$ in step 6 of the PI extraction algorithm, a table of pointers $\{i_j, 0 \leq j < 2^k\}$ must be generated which indexes elements of the subspace $V_b$ as a subset of $\mathbb{Z}^n$. The following algorithm requires approximately $2^k$ indexed additions to generate this table. The same principle can be used to generate $|x|$, $|x|$ mod 2, or similar quantities.

The first step is to identify the set of weights $\{2^{n-r} : b_r = 0\}$ whose sums (over subsets) define the integers $i_j$ whose radix-two expansions are elements of $V_b$. Let the $j$th power of two in ascending order from this set be $s_j = 2^j$, $1 \leq j \leq k$. Then the $2^k$-dimensional vector $P_k$ which contains the ordered list of pointers $\{i_j : x(i_j) \in V_b\}$ is defined by the recursion relation

$$P_1 = (0, s_1)$$

$$P_{j+1} = (1, 1) \times P_j + (0, 1) \times [s_{j+1} \cdot E_j], \quad \text{for } 1 \leq j \leq k \quad (86)$$

Here $\times$ denotes the Kronecker product (see Section II.A.4), and $E_j$ denotes a constant row vector containing $2^j$ unit entries.

The above formula may be verified directly. It states that the first $2^j$ entries of $P_{j+1}$ and $P_j$ are identical, while the last $2^j$ entries of $P_{j+1}$ are obtained by adding $s_{j+1}$ to corresponding entries of $P_j$:

$$P_{j+1}(2^j + i) = P_j(i) + s_{j+1} \quad \text{for } 0 \leq i < 2^j \quad (87)$$

3. Rejection of Nonprime Implicants

Step (4) of the above algorithm detects all implicants of $f$ and $\bar{f}$ but does not identify prime implicants (those covered by larger ones). The following simple test to reject nonprime implicants is an adaptation of the "nearest neighbor" or "adjacent vertex" test used in recent algorithms for the direct buildup of larger implicants out of smaller ones (Morreale, 1967; Necula, 1967).

If a coset $V_b + c$ is included in a larger one, then the larger one must include all the free variables of $V_b + c$, plus at least one more free variable that was a bound variable in $V_b + c$. Suppose $x_i$ was bound in $V_b + c$ but is free in this larger implicant; then the coset $V_b + (c \oplus e_i)$ is also an implicant of $f$ or $\bar{f}$ ($e_i$ is the unit vector corresponding to $x_i$, and the vector sum $c \oplus e_i$ is modulo two). Let $c = x(i_j)$ correspond to the argument $y(j)$ of $(f^* \cap V_b)^*$. The above observation is formalized in the following theorem:
**Theorem 3.1.** Let \((f^* \Delta V_b)^* = 2^n\) or 0 at the argument \(y(j)\) corresponding to \(c(j) = x(i_j)\) in \(Z^n\). Then the coset \(V_b + c(j)\) is a prime implicant iff \((f^* \Delta V_b)^*\) does not have the same value (2\(^n\) or 0 respectively) at any of the \(k\) adjacent arguments \((y \oplus e_p), 1 \leq p \leq k\) (\(e_p\) is the \(p\)th unit vector in \(Z^k\)).

If the test described above indicates that \(V_b + c\) is redundant, then at least one other coset is also known to be redundant. If the second redundant coset is tagged, a duplicate test for redundancy can be avoided when the latter coset is scanned. The average number of tests (each requiring one operation to find the pointer \((i_j \pm 2^{k-1}) = q\) and another to test \((f^* \Delta V_b)^*\) at the argument \(y(q)\)) is reduced from \(k\) to \(k/2\) per implicant.

4. Identification of Essential Prime Implicants

An essential or "core" prime implicant (PI) of \(f(x)\) is one which covers some point of \(f(x)\) that is not covered by any other PI. Every minimal normal form which covers \(f\) must include all of the essential PI's. Although the PI extraction algorithm does not identify the essential PI's of \(f\) and \(f\), it does provide an efficient means of computing the integer-valued function \(g(x)\)

\[
g(x) = \sum \text{(PI's of } f) - \sum \text{(PI's of } f')
\]

(88)

If this function is 1 or \(-1\) at a particular point, it means that only one PI of \(f\) or \(f'\) respectively, covers the point \(x\).

The algorithm computes \(g^*\), the transform of \(g\), rather than \(g\) itself (2\(^n\) storage locations are required). Linearity permits the accumulation of \(g^*\) as a sum (over all \(b\)) of transforms \(\Delta g_b^*(w)\), where \(\Delta g_b(x)\) is the contribution to \(g\) of all PI's which are cosets of the same subspace \(V_b\). The cost of adding \((\Delta g_b(x))^*\) to \(g^*\) is merely \((k + 3)2^k\), which (for small \(k\) at least) is much less than the \(m2^{n-k}\) operations required to compute \(\Delta g_b\) directly. (The break-even point depends on the number \(m\) of PI's that are cosets of \(V_b\).) Suppose that \(m\) PI's of \(f\) or \(f'\) have been identified among the cosets of \(V_b\). Define \(v_{b,c_i}(x)\) as the characteristic function of the \(i\)th prime implicant \(V_b + c_i\) (i.e., \(v_{c_i}(1) = V_b + c_i\), and suppose also that \(V_b + c_i \subseteq f^{-1}(x)\); for \(1 \leq i \leq m\). Define \(\Delta g_b(x)\) as the quantity contributed to the function \(g(x)\) by cosets of \(V_b\):

\[
\Delta g_b(x) = \sum_{i=1}^{m} (-1)^{1+z_i} v_{b,c_i}(x) \quad \text{(real sum)}
\]

(89)

By linearity of the Fourier transform operation, it follows that

\[
(g(x) + \Delta g_b(x))^* = g^*(w) + \Delta g_b^*(w) \quad \text{(real sum)}
\]

(90)

The following theorem defines the transform of \(\Delta g_b\):
**THEOREM 3.2.** Let

\[ \Delta g_b(x) = \sum_{i=1}^{m} (-1)^{1+x_i} p_{b,c_i}(x) \]

on \( \mathbb{Z}^n \) and let

\[ h(x) = \sum_{i=1}^{m} (-1)^{1+x_i} \delta_{x,c_i}, \quad \text{for} \quad x \in V_b \]

Then, if \( w = w_1 \oplus w_2 \), with \( w_1 \in V_b \) and \( w_2 \in V_b \),

\[ \Delta g_b^*(w) = 2^{n-k} \delta_{w_2,0} h^*(w_1) \tag{91} \]

where \( n - k = |b| \) is the dimension of \( V_b \).

**Proof:** By its definition, \( \Delta g_b(x) \) is constant on cosets of \( V_b \); therefore, the quotient group character theorem of Section II,B,2 applies:

\[ \Delta g_b^*(w) = 2^{n-k} \delta_{w_2,0} (\Delta g_b(x) \cap V_b)^* \tag{92} \]

The restriction of \( \Delta g_b \) to \( V_b \) is merely \( h(x) \).

To apply this theorem, first define \( h(y) = (-1)^{1+y_i} \) for \( y(f) \) corresponding to the coset leader \( c(i_j) \) of each PI \( (h(y) = 0 \) elsewhere on \( \mathbb{Z}^k \)). Next compute \( h^* \) on \( \mathbb{Z}^k \) and finally use the table of subspace pointers \( \{i_j, 0 \leq j < 2^k\} \) to add \( h^*(y(j)) \) to \( g^*(x(i_j)) \). This accomplishes step (6) of the PI extraction algorithm. Note that \( \delta_{w,0} \) implies \( \Delta g_b^*(w) = 0 \) except on \( V_b \).

**Example 14.** The contribution \( \Delta g(x) \) of the three PI’s detected in the example of Section III,A,4 will be computed. The function \( h(y) \) identifies two implicants of \( f \) and one of \( f' \):

\[ h(y) = (1, 0, 0, 1, 0, 0, 0, -1) \tag{93} \]

Using the matrix notation of Fig. 4, \( \Delta g^* \cap V_b = 2^{n-k} h Q^{[3]} = (12, -4, -4, 12, 4, 4, 4, 4) \). The entire \( 4 \times 8 \) array \( \Delta G^* \) representing \( \Delta g^*(x) \) is merely the product \((1, 0, 0, 0)^t(\Delta g^* \cap V_b)\).

To take the inverse transform, merely premultiply by \( Q^{[2]} \), postmultiply by \( Q^{[3]} \), and multiply by \( 2^{-5} \):

\[ Q^{[2]}(1, 0, 0, 0)^t = (1, 1, 1, 1)^t \tag{94} \]

and

\[ 2^{-5}(\Delta g^* \cap V_b)Q^{[3]} = (1, 0, 0, 1, 0, 0, 0, -1) \tag{95} \]
The product of these two arrays is

\[ G = 2^{-5}Q^{[3]}(\Delta G^*)Q^{[3]} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix} \] (96)

This matrix accurately represents the number of PI's that cover each element \( x \) of \( Z^5 \).

5. Computational Bounds

This section will derive an accurate bound on the number of elementary (indexed add or subtract) operations required to extract prime implicents and compare it with more direct approaches. Consider a subspace \( V_b \) of dimension \( n - k \), whose \( 2^k \) cosets \( V_b + c \) are to be tested to see which of them are \( k \)-literal implicents of \( f \). \( V_b \) has dimension \( k \), and the restriction of \( f^* \) to \( V_b \) has \( 2^k \) elements. Computation of the inverse transform \( f^{**} \) of \( (f^* \cap V_b) \) requires \( k2^{k+1} \) operations; \( 2^k \) additions are required to compute pointers to the subset \( V_b \subseteq Z^n \); \( 2^k \) operations are required to extract the \( 2^k \) elements of \( f^* \) on \( V_b \), and \( 2^k \) more are required to test \( f^{**} \) for each coset leader. To verify "primeness," an average of \( k \) additional operations are required for each PI. A total of \( (3k + 3)2^k \) operations are required to identify all PI's corresponding to cosets of \( V_b \), if we assume the test for primeness is carried out for every one of the cosets \( V_b + c \).

There are \( 2^n \) subspaces of \( Z^n \), \( C_{n,k} = n!/((k!)(n-k)!)) \) of which have dimension \( k \). Summing over all of these subspaces gives the following bound on total computation time to test all subcubes of \( Z^n \):

\[
\sum_{k=1}^{n-1} C_{n,k} 3(k+1)2^k = 3^r(2n + 3)
\] (97)

or an average of \( 2n + 3 \) operations for every subcube of \( Z^n \). This bound grows at a rate \( 2n + 3 \) times faster than \( 3^r \), the total number of subcubes.

The exponential growth of this function, and its low cost for small values of \( k \), suggests the possibility that another, direct approach might be more efficient after \( k \) passes some threshold. This possibility has been explored.

The two basic approaches to extraction of prime implicents in the literature involve comparison of each minterm \( x \) in \( f^{-1}(1) \) or \( f^{-1}(0) \) with either all of its adjacent vertices in \( f^{-1}(1) \) or \( f^{-1}(0) \) (Quine, 1955; McCluskey, 1956) or with all \( n \) adjacent vertices on the \( n \)-cube (Morreale, 1967; Necula, 1967). Knowledge about large subcubes of \( f^{-1}(1) \) is built up gradually. The computational burden grows roughly as the square of the number of minterms in \( f^{-1}(1) \) or \( f^{-1}(0) \) and (generally speaking) prime implicents are only extracted
for the smaller of these two sets. Since realistic bounds on computation time for these algorithms are difficult to obtain, exhaustive search was postulated as a reference procedure, similar to the direct summation over cosets that was illustrated in Fig. 4. Comparative results are shown in Table I.

**TABLE I**

Computation Time (Millions of Operations)

<table>
<thead>
<tr>
<th>N</th>
<th>K</th>
<th>DIRECT</th>
<th>TRANSFORM</th>
<th>OLD/NEW</th>
<th>SUMOLD</th>
<th>SUMNEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>5.313</td>
<td>0.0003</td>
<td>7.0001</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>1.778</td>
<td>0.0004</td>
<td>7.0004</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.667</td>
<td>0.0015</td>
<td>7.0014</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>0.267</td>
<td>0.0019</td>
<td>7.0026</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.02</td>
<td>0.00</td>
<td>170.667</td>
<td>0.0225</td>
<td>7.0001</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0.09</td>
<td>0.00</td>
<td>56.889</td>
<td>0.1124</td>
<td>7.0017</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>0.25</td>
<td>0.01</td>
<td>21.333</td>
<td>0.3584</td>
<td>7.0033</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>0.43</td>
<td>0.05</td>
<td>8.533</td>
<td>0.7885</td>
<td>7.0063</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>0.52</td>
<td>0.15</td>
<td>3.556</td>
<td>1.3046</td>
<td>7.0088</td>
</tr>
<tr>
<td>10</td>
<td>6</td>
<td>0.43</td>
<td>0.28</td>
<td>1.524</td>
<td>1.7947</td>
<td>7.0131</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>0.25</td>
<td>0.37</td>
<td>0.667</td>
<td>1.9804</td>
<td>7.0397</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>0.09</td>
<td>0.31</td>
<td>0.296</td>
<td>2.0726</td>
<td>7.1707</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>0.02</td>
<td>0.15</td>
<td>0.133</td>
<td>2.0931</td>
<td>7.3243</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>0.48</td>
<td>0.00</td>
<td>5461.332</td>
<td>7.9830</td>
<td>7.0002</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>6.88</td>
<td>0.00</td>
<td>1820.444</td>
<td>7.8643</td>
<td>7.0040</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>20.82</td>
<td>0.04</td>
<td>582.667</td>
<td>37.6531</td>
<td>7.0047</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>69.46</td>
<td>0.33</td>
<td>273.067</td>
<td>127.1337</td>
<td>7.0375</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>196.80</td>
<td>1.73</td>
<td>113.778</td>
<td>323.6938</td>
<td>7.1050</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>328.01</td>
<td>6.73</td>
<td>48.762</td>
<td>651.0500</td>
<td>8.8317</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>421.72</td>
<td>19.77</td>
<td>21.333</td>
<td>1073.6750</td>
<td>26.5999</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>421.72</td>
<td>44.48</td>
<td>9.481</td>
<td>1405.3980</td>
<td>71.0786</td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>328.01</td>
<td>76.28</td>
<td>4.267</td>
<td>1823.4060</td>
<td>149.5533</td>
</tr>
<tr>
<td>15</td>
<td>10</td>
<td>196.80</td>
<td>101.18</td>
<td>1.939</td>
<td>2020.2100</td>
<td>251.4325</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>89.46</td>
<td>106.16</td>
<td>0.889</td>
<td>2109.6660</td>
<td>352.0708</td>
</tr>
<tr>
<td>15</td>
<td>12</td>
<td>29.20</td>
<td>72.68</td>
<td>0.417</td>
<td>2120.4850</td>
<td>829.7502</td>
</tr>
<tr>
<td>15</td>
<td>13</td>
<td>6.08</td>
<td>36.13</td>
<td>0.190</td>
<td>2146.3660</td>
<td>460.8806</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>0.08</td>
<td>11.06</td>
<td>0.089</td>
<td>2187.3490</td>
<td>471.9395</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>81.04</td>
<td>0.00</td>
<td>11024.220</td>
<td>440.4014</td>
<td>7.0002</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>388.46</td>
<td>0.01</td>
<td>58254.220</td>
<td>440.4014</td>
<td>7.0071</td>
</tr>
<tr>
<td>20</td>
<td>3</td>
<td>2390.75</td>
<td>0.11</td>
<td>21845.339</td>
<td>2831.1520</td>
<td>7.0115</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>8738.133</td>
<td>1.16</td>
<td>8738.133</td>
<td>2831.1520</td>
<td>1.2793</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>3640.869</td>
<td>8.93</td>
<td>3640.869</td>
<td>1560.381</td>
<td>10.2096</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>1560.381</td>
<td>52.09</td>
<td>1560.381</td>
<td>62.3029</td>
<td>10.2096</td>
</tr>
<tr>
<td>20</td>
<td>7</td>
<td>238.14</td>
<td>238.14</td>
<td>682.667</td>
<td>300.4436</td>
<td>1171.1460</td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>870.70</td>
<td>870.70</td>
<td>303.407</td>
<td>136.533</td>
<td>3751.0950</td>
</tr>
<tr>
<td>20</td>
<td>9</td>
<td>2579.86</td>
<td>2579.86</td>
<td>136.533</td>
<td>3751.0950</td>
<td>1171.1460</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>6243.25</td>
<td>6243.25</td>
<td>62.061</td>
<td>099.260</td>
<td>19099.060</td>
</tr>
<tr>
<td>20</td>
<td>12</td>
<td>13.128</td>
<td>13.128</td>
<td>13.128</td>
<td>13.128</td>
<td>1171.1460</td>
</tr>
<tr>
<td>20</td>
<td>13</td>
<td>6.005</td>
<td>6.005</td>
<td>6.005</td>
<td>6.005</td>
<td>1171.1460</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>2.844</td>
<td>2.844</td>
<td>2.844</td>
<td>2.844</td>
<td>1171.1460</td>
</tr>
<tr>
<td>20</td>
<td>15</td>
<td>1.333</td>
<td>1.333</td>
<td>1.333</td>
<td>1.333</td>
<td>1171.1460</td>
</tr>
<tr>
<td>20</td>
<td>16</td>
<td>0.627</td>
<td>0.627</td>
<td>0.627</td>
<td>0.627</td>
<td>1171.1460</td>
</tr>
<tr>
<td>20</td>
<td>17</td>
<td>2300.75</td>
<td>2300.75</td>
<td>0.206</td>
<td>0.206</td>
<td>1171.1460</td>
</tr>
<tr>
<td>20</td>
<td>18</td>
<td>398.46</td>
<td>2383.02</td>
<td>0.140</td>
<td>0.140</td>
<td>1171.1460</td>
</tr>
<tr>
<td>20</td>
<td>19</td>
<td>0.007</td>
<td>0.007</td>
<td>0.007</td>
<td>0.007</td>
<td>1171.1460</td>
</tr>
</tbody>
</table>
To compute \( \sum \{ f(x) : x \in V_b + c \} \) requires \( 2^{n-k} \) pointers to each of \( 2^k \) cosets of \( V_b \) and \( (2^{n-k}) \) table lookups to sum \( f(x) \) over each of the \( 2^k \) cosets. The total cost for each \( k \)-dimensional subspace is \( 2^{n+1} \). Multiplying by \( C_{n,k} \) and summing over \( k \) gives a total of \( 2(2^n)^2 = 2(4)^n \) operations, or an average of \( 2(\frac{3}{4})^n > 2(3)^{n/4} \) operations for each of the \( 3^n \) subcubes of \( \mathbb{Z}^n \), rather than \( 2n + 3 \) as in our algorithm.

The incremental cost for each \( k \) and the cumulative sum of each cost as \( k \) increases from 1 to \( n - 1 \) is shown for \( n = 5, 10, 15, 20 \) in Table I. The lowest value of \( k \) for which exhaustive search has a lower cost than the method herein is \( k = 7 \) for \( n = 10, k = 11 \), for \( n = 15 \), and \( k = 16 \) for \( n = 20 \). For \( n = 15 \) or 20, the cumulative saving by reverting to direct search for \( k \geq 11 \) or 16, respectively, is about 30% of the total time for implicant computation. At 500,000 indexed additions per second, the total time for \( n = 15 \) is 16 min for the transform algorithm, 11 min. for the hybrid (transform approach for \( k \leq 10 \), direct search for \( k > 10 \)) and 75 min if direct search is used for all \( k \).

6. Storage Bounds

The fact that all PI’s need not be stored in core memory produces a very significant reduction in storage compared to previous methods. Meo (1968) observes that functions of ten or more variables may have many more than \( 2^n \) PI’s. He estimated the size of memory required to compute the PI’s of a function of \( n \) variables by either the Quine–McCluskey method (McCluskey, 1956) or the consensus rule (Quine, 1955). This size is half the maximum number of subcubes of any single dimension \( (n-k) \), \( C_M = \frac{1}{2} \max \{ C_{n,k}2^{n-k}, 1 \leq k < n \} \). In contrast, our method requires \( 2^n \) terms in the array \( f^* \) plus \( 2^{n-1} \) terms in \( (f^* \setminus V_b) \). Data from which to determine essential PI’s doubles this to \( 3 \cdot 2^n \) locations. A comparison of \( C_M \) and \( 3 \cdot 2^n \) is given in Table II.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( C_M )</th>
<th>( 3 \cdot 2^n )</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>40</td>
<td>96</td>
<td>1/2</td>
</tr>
<tr>
<td>10</td>
<td>7,680</td>
<td>3,072</td>
<td>2/1</td>
</tr>
<tr>
<td>15</td>
<td>1,540,000</td>
<td>98,704</td>
<td>15/1</td>
</tr>
<tr>
<td>20</td>
<td>3,175,000</td>
<td>1,048,566</td>
<td>300/1</td>
</tr>
</tbody>
</table>

The bound \( C_M \) grows by a factor of about 3 for each increment of \( n \); the bound \( 3 \cdot 2^n \) doubles with each increment of \( n \).
7. Functions with "Don't Care" Conditions

The preceding algorithm will now be extended to functions that are undefined for certain values of $x$ in $\mathbb{Z}^n$. The extension is straightforward, but the symmetry that permitted simultaneous extraction of the implicans of both $f$ and $\bar{f}$ is not possible. Suppose "-" denotes the set $(0, 1)$ of range elements into which each don't care argument state is mapped. Then the don't care set $f^{-1}(-)$ is defined along with $f^{-1}(1)$ and $f^{-1}(0)$. To obtain the prime implicants of $f$, we apply the algorithm just defined to the transform of the function $g(x)$ such that $g^{-1}(1) = f^{-1}(1) \cup f^{-1}(-) = K$; to determine implicants of $\bar{f}$, we must begin with the transform of $h(x)$ such that $h^{-1}(1) = f^{-1}(1) = L$. Since these are transforms of two different functions, the entire algorithm except for steps (1) and (6) must be repeated to get PI's of $\bar{f}$ as well as $f$. Instead of $3(k+1)2^k$, the incremental cost for each subspace becomes $5(k+1)2^k$, an increase of 60%. However, simultaneous computation of PI's for $f$ and $\bar{f}$ will detect every coset that is an implicant of both sets $f^{-1}(1) \cup f^{-1}(-)$ and $f^{-1}(0) \cup f^{-1}(-)$, i.e., it is an implicant of $f^{-1}(-)$ only. Such implicants are redundant in any covering of either $f^{-1}(1)$ or $f^{-1}(0)$, and should be discarded.

C. TEST FOR DISJUNCTIVE DECOMPOSITION

A test for disjunctive decomposition is a byproduct of the transform approach to implicant extraction. This test relies on the property that

$$ (f^* \land V_b)^* = \sum \{ f(x) : x \in V_b + c \} $$

and

$$ (f^* \land V_b)^* = \sum \{ f(x) : x \in V_b + c \}. $$

Ashenhurst (1959) has shown that a function which possesses a disjunctive decomposition $f(x) = g(z, \varphi(y))$, with $y = x \land V_b$ and $z = x \land V_b$, has an easily recognized pattern to its truth table when the latter is shown on the cartesian product of the subspaces $V_b$ and $V_b$ (as in Fig. 4). This pattern arises because a decomposable function must either be constant at 0 or 1, equal to $\varphi(y)$, or equal to $\varphi(y)$ on a particular coset of $V_b$. In other words, every column sum over cosets of $V_b$ must have at most two distinct values between 0 and $2^n - k$. This test is easily implemented by keeping track of the number of distinct coefficient values possessed by the inverse transform $(f^* \land V_b)^* 2^{k-n}$ during the implicant extraction algorithm.

Another necessary condition for functional decomposition can be derived from the row sums, which are obtained when cosets of $V_b$ are processed by the PI extraction algorithm. A decomposable function of the form $f(x) = g(z, \varphi(y))$, $y = x \land V_b$, $z = x \land V_b$ can have at most four distinct row sums
over cosets of $V_b$. Suppose that $f \cap V_b + z$ is a constant, independent of $y$; if $f = 0$ or 1, the row sum is 0 or $2^k$, respectively; the only other possibilities are that $f \cap V_b + z = \phi(y)$ or $f \cap V_b + z = \phi(y')$. The row sums in these two cases are $|\phi|$ and $2^k - |\phi|$, where $|\phi|$ is the number of unit entries in the truth table for $\phi(y)$, and $2^k - |\phi|$ is the corresponding quantity for $\phi$.

The above two tests furnish two sufficient conditions that in practice may discriminate against a large fraction of the $2^n - 1$ possible partitions of $Z^a$ into two subspaces $V_b$ and $V_b$. Both tests should be applied to each subspace $V_b$ as it is processed. A two-bit tag must be reserved for each of the $2^n$ possible vectors $b$ to save the results of both tests for the subspace $V_b$ so that later on when $V_b$ is processed, both necessary conditions can be verified. The partition of variables for the decomposition, and in fact the truth table array on $V_b \times V_b$ can then be printed out or otherwise used (e.g., to extract implicants of $g(z, \phi)$ and $\phi(y)$ rather than $f(x)$).

The advantage of this procedure depends directly on the fraction of the $2^n$ partitions of variables which are rejected by the test without requiring $2^n$ row and column sums or other pattern recognition techniques to verify sufficient conditions for a true decomposition.

A new test for disjunctive decomposition was described by Shen et al. (1969). This test is extremely efficient for randomly chosen functions but slow for functions which actually do possess a decomposition. The test herein is probably less efficient (for random functions) when performed alone; however, when performed during prime implicant extraction, it probably involves negligible labor compared to Shen's method.

**D. CRITERIA FOR FACTORING VARIABLES**

When a function has too many arguments to realize directly and does not possess a decomposition, then some (say, $n - k$) of the variables can be factored out and used to address a canonical decoding tree which selects one of $2^{n-k}$ functions of the remaining $k$ arguments to be evaluated. Actually, mixed factoring can be used; for example, in the expression

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1(x_2 g_1(x_3, x_4) \lor \bar{x}_2 g_0(x_3, x_4))$$

$$\lor \ x_3 h_1(x_2, x_4) \lor \bar{x}_3 h_0(x_2, x_4))$$

In this expression, the choice of the second variable to be factored depends on whether $x_1 = 0$ or 1. In general, there are $n!$ ways to order the variables and many more ways to reorder them after some of them have been decoded.

To avoid exhaustive procedures, some complexity criterion is desirable by
which to rank the variables. One such criterion is provided by the various sums computed by the PI extraction algorithm. For example, for \( k = 1 \), there are five subspaces \( V_k \), and \((f^* \setminus V_k)^*\) identifies the sum \( \sum \{ f(x) : x \in V_k + c \} \) for each subspace. Now, generally speaking (but not always), the closer the sum of \( f(x) \) over \( V_k + c \) is to 0 or \( 2^{n-k} \), the easier it is to construct the function. Therefore, the variable corresponding to the 1-dimensional subspace \( V_1 \), which produces the largest or smallest value of \((f^* \setminus V_1)^* = \sum \{ f(x) : x \in V_1 + c \} \) is the most promising candidate for initial factorization according to this heuristic criterion.

After \( x_i \) has been factored out, it is easy to rearrange \( f^* \) to separate \( f^* \setminus V_k \) from \( f^* \setminus V_i + c \), so that each can be processed separately. The preceding criterion can be applied on each subfunction separately to choose the next variable to be factored (independently for each subfunction).

**EXERCISES**

8. Find the characteristic functions of the subspaces of \( \mathbb{Z}^5 \) defined by the vectors \( b_1 = (00111), b_2 = (01010), b_3 = (10000) \). That is, find

\[
v_k^{-1}(1) = \{ i : x(i) \in V_k \}
\]

9. Find the nullspace for each subspace \( V_i \) in the preceding problem; define each nullspace by its ternary \( k \)-cube symbol.

10. Compute \((f^* \setminus V_1)^*\) for the 3-argument function \( f(x) = x_1 x_2 \lor x_3 \) and the two subspaces defined by \( b_1 = (001), b_2 = (110) \). Show that \( x_1, x_2 \) and \( x_3 \) are prime implicants of \( f \) by applying the PI extraction theorem to the result.

11. Show that the function \( f^{-1}(1) = \{ 0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11 \} \) has a disjunctive decomposition of the form \( f = g(x_1, x_3, h(x_3, x_4)) \) by applying the test of Section III.C.

**IV. ANALYSIS OF THE PROTOTYPE EQUIVALENCE RELATION**

This section defines the restricted affine group RAG, a subgroup of the symmetric group on the cartesian product or direct sum \( X + \mathbb{Z} \) of the vector space \( X = \mathbb{Z}^n \) of arguments of a combinational logic function and the vector
space \( Z \) of its outputs. Elements of this space are \((n + 1)\)-tuples \((x, z)\) with \( z = f(x) \). The group RAG permutes this space under restrictions which preserve the single-valuedness of all functions from \( X \) into \( Z \). The equivalence relation induced on the space \( \mathcal{F} \) of all such functions is called the \textit{prototype} equivalence relation, after Ninomiya (1958), and the equivalence classes into which \( \mathcal{F} \) is partitioned by RAG are called \textit{prototype classes}. The group RAG is the largest subgroup of the affine group on \( X + Z \) that retains the loop-free or feedback-free constraint which prevents combinational logic from becoming asynchronous sequential logic circuitry.

Section IV,A gives a mathematical definition and physical interpretation of elements of RAG and locates each transformation group previously applied to combinational logic within a lattice of major subgroups of RAG. It also explicitly defines the weight-transforming property which distinguishes RAG from permutation groups acting on \( X \) alone.

Section IV,B gives results on the number of equivalence classes of functions under RAG for \( n \leq 5 \). For the case \( n = 3 \), an explicit breakdown is given of prototype classes into generic classes (under argument permutation, argument, and/or function complementation). More detailed results are also provided on the number of classes of functions with a specific value of their weight or size of \( f^{-1}(1) \), under linear and affine groups acting on \( X \).

Section IV,C contains explicit data on 46 out of the 48 possible 5-variable prototypes. These 46 classes were obtained by a nonexhaustive method, and the remaining ambiguity has not been resolved.

\[ A. \text{ DEFINITION, INTERPRETATION, AND CONTEXT} \]

In this section, the prototype equivalence relation is interpreted physically, defined algebraically, and placed within the historical context of other equivalence relations of interest in switching theory.

\[ 1. \text{ Notation} \]

The notation at the beginning of Section II,A will be used. The full linear group and the full affine group on \( Z^n \) will be denoted \( \text{LG}(n) \) and \( \text{AG}(n) \), respectively. Elements of \( \text{LG}(n) \) and \( \text{AG}(n) \) will be represented as follows (Birkhoff and MacLane, 1953):

\[
T_A(x) = xA \text{ is in } \text{LG}(n) \text{ iff } A \text{ nonsingular over } Z
\]

\[ T_{A,c}(x) = xA \oplus c \text{ is in } \text{AG}(n) \text{ iff } T_A \text{ is in } \text{LG}(n) \text{ and } c \in Z \quad (102) \]

The direct product of \( \text{AG}(n) \) and \( \text{AG}(m) \) is denoted \( \text{AG}(n) \times \text{AG}(m) \) and similarly with \( \text{LG}(n) \times \text{LG}(m) \).
2. Engineering Motivation and Definition

In order to motivate our definition of prototype equivalence, some engineering considerations will be introduced. Consider first the group AG(n + 1) of all affine transformations $T_{\tilde{\alpha}e}$ which operate on vectors $z$ in $\mathbb{Z}^{n+1}$ by post-multiplication: $zT_{\tilde{\alpha}e} = z\tilde{\alpha} \oplus \tilde{e}$. We represent these transformations as matrices which operate on ordered pairs $(x, z)$ where $x$ is the input to, and $z = f(x)$ is the output of, an element of $\mathcal{F}$. To show the effects of $T_{\tilde{\alpha}e}$ on $x$ and $z$ explicitly, we partition the matrix $\tilde{\alpha}$ and translation vector $\tilde{c}$:

$$T(x; z) = (x; z) \begin{bmatrix} A & a^t \\ b & e \end{bmatrix} \oplus (c; d) = (xA \oplus zb \oplus c; ze \oplus xa^t \oplus d)$$

(103)

Note that $a$, $b$, and $c$ are binary $n$-tuples, while $d$ and $e$ are binary scalars; furthermore, $A$ can be singular and $e$ can be zero as long as the complete matrix is nonsingular.

If $a$ and $b$ are restricted to be identically zero (and $e = 1$) to retain non-singularity in Eq. (103), we obtain a representation for the direct product group $\text{AG}(n) \times \text{AG}(1)$ which includes all the subgroups previously treated in the literature of switching theory. The restricted affine group RAG also requires $b$ to be zero and $e = 1$, but $a$ may range over all of $\mathbb{Z}^n$. In other words, the following proper inclusion relation applies:

$$\text{AG}(n) \times \text{AG}(1) \subset \text{RAG} \subset \text{AG}(n + 1)$$

(104)

Our definition of RAG will be motivated by first studying the effects of AG(n + 1) and AG(n) × AG(1).

The direct product of two affine groups acting on $X$ and $Z$, respectively, produces mutually independent encodings on the domain $X$ and the range $Z$ of a switching function, as illustrated in Fig. 5a. Since AG(1) has only two elements, range transformations at most double the size of the transformation group and reduce the number of equivalence classes by a factor of at most two. However, for multiple-output functions, $Z$ becomes $\mathbb{Z}^m$, AG(n) becomes AG(m), and such a direct product extension is nontrivial.

Figure 5a gives a physical interpretation of the condition for equivalence of two functions $f$ and $g$ under the group AG(n) × AG(1). The domain encoding is a transformation of the form $y = xA \oplus c$, $A$ nonsingular over $Z$ and $c$ arbitrary in $\mathbb{Z}^n$. With the exception of Ninomiya (1958) and Stone and Jackson (1966), the literature on functional equivalence under transformation groups is devoted exclusively to direct product groups with a subgroup of AG(n) acting on $X$ and a subgroup of AG(m) acting independently on the range.

Figure 5b illustrates why the full affine group AG(n + 1) is not a proper object for study. The value of $y = xA \oplus zb \oplus c$ in Eq. (103) depends on the
FIG. 5. Models for single-valued functions imbedded in encoding transformations on \(X + Z\). (a) Fixed domain and range encodings. (b) Mutually-interacting domain and range encodings. (c) Input-dependent range encoding; no feedback.

function output \(z\). Unfortunately, adding \(zb\) to the function input implies ambiguities and possible instabilities usually associated with uncontrolled feedback loops within combinational logic circuits. In sequential circuits, feedback is controlled by inserting delays in the feedback path. In our (combinational) logic model, feedback must be avoided by forcing \(b\) to be zero in Eq. (103).

On the other hand, feed-forward of the sum \(xa^i\) can still be used to modify the output of an arbitrary switching function. This does not violate our restriction to feedback-free combinational logic circuits, yet it generalizes the direct product group \(AG(n) \times AG(1)\) by introducing \(n\) more free variables (\(a_1\) through \(a_n\)). This intermediate case is illustrated in Fig. 5c. In this model, an affine transformation operates on the domain or space of inputs \(x\) to produce the (encoded) input \(y = xA \oplus c\) to the logic circuit. Its output \(g(y)\) together with the function arguments \(x_1, x_2, \ldots, x_n\) are linearly combined by a range transformation which defines \(f(x)\) in the following way:

\[
f(x) = g(y) \oplus x_1a_1 \oplus \cdots \oplus x_n a_n \oplus d = g(xA \oplus c) \oplus xa^i \oplus d
\]

where \(d\) and each \(a_i\) are binary constants. In effect, we have produced \(f(x)\) by adding a linear polynomial \(d \oplus xa^i\) to the output of a logic circuit \(g\) whose
input $y$ was defined by an affine transformation of $x$. The complete expression for $f(x)$ is

$$f(x) = g(xA \oplus c) \oplus xa^i \oplus d$$

(105)

In matrix form,

$$(y, g) = (x, f) \begin{bmatrix} A & a^i \\ 0 & 1 \end{bmatrix} \oplus (c, d)$$

(106)

(Eq. (103) with $b = 0$).

**Definition.** The Restricted Affine Group on $X + Z$ is that subgroup of affine transformations on $X + Z$ whose partitioned matrix forms obey the constraint $b = 0$.

That RAG is actually a subgroup can be verified by direct multiplication. A more rigorous characterization of RAG will be given in a later section (in terms of invariant subspace restrictions). The name RAG was given to this group by Lechner (1963a,b) (see Section A.4). The group RAG was first studied by Ninomiya (1958), who gave the name “prototype equivalence” to the relation it induces on the function space $\mathcal{F}$.

The term “restricted affine equivalence” is also reasonable for the equivalence relation induced by RAG as a subgroup of $AG(n + 1)$ (although “generalized” affine equivalence is just as valid if we regard RAG as a group which includes the direct product $AG(n) \times AG(1)$). However, we will retain Ninomiya’s term “prototype equivalence,” not only because there is little likelihood of applying larger transformation groups to $\mathcal{F}$, but also to avoid possible confusion with the recent use of the term “affine equivalence” to refer to the subgroup of $AG(n) \times AG(1)$ which induces Ninomiya’s “family” equivalence relation (Stone and Jackson, 1969). In this subgroup, $A$ is restricted to be a permutation matrix, and “affine equivalence” without further qualification has a broader connotation than the family relation.

The reader who desires additional engineering content can now skip to Section V, which describes encoded-input logic, a new logic synthesis technique based on the group RAG. In this approach, the additions implied by Eq. (106) are actually performed by a physical array of cells containing 2-input, single-output EXCLUSIVE OR operators, one for each unit component of $A$, $a$, $c$, and $d$. In this way, any function $f$ which is equivalent to $g$ under the group RAG can be realized by embedding $g$ within the proper encoding transformation (see Fig. 5c).

The group RAG is actually a special case of a broad class of affine subgroups which will be defined in a later section. The zero submatrix restriction ($b = 0$) will be generalized and related to invariant subspace restrictions when such a group acts on $Z^{n+1}$. 
3. Lattice of Classical Subgroups

The group RAG includes as subgroups most of the transformation groups whose induced equivalence relations have been studied in the literature of switching theory. Figure 6 illustrates the partial ordering among the subgroups of RAG. Table III defines each subgroup by specific restrictions on A, a, c, and d. Bibliographic references to enumeration of equivalence classes under these groups are given in the text.

The two left hand columns of this lattice involve transformation groups on the domain Χ alone. The function output is left untouched. The first column includes the group of input permutations and its generalization to the full linear group (Hellerman, 1961; Harrison, 1964; Slepian, 1960). Slepian's results on group code equivalence under LG(n) have also been translated into switching theory terminology (Lechner, 1967).

The lowest point in the second column is the group that complements
TABLE III
Restrictions on RAG to Obtain Subgroups

<table>
<thead>
<tr>
<th>Subgroup or Equivalence Relation</th>
<th>A</th>
<th>c</th>
<th>a</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype</td>
<td>A</td>
<td>c</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>Family</td>
<td>P</td>
<td>c</td>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>Add Linear Functions</td>
<td>I</td>
<td>g</td>
<td>a</td>
<td>g</td>
</tr>
<tr>
<td>AG(n) x AG(1)</td>
<td>A</td>
<td>c</td>
<td>g</td>
<td>d</td>
</tr>
<tr>
<td>Genus</td>
<td>P</td>
<td>c</td>
<td>g</td>
<td>d</td>
</tr>
<tr>
<td>Complement Function</td>
<td>I</td>
<td>g</td>
<td>g</td>
<td>d</td>
</tr>
<tr>
<td>AG(n)</td>
<td>A</td>
<td>c</td>
<td>g</td>
<td>g</td>
</tr>
<tr>
<td>Symmetry Types</td>
<td>P</td>
<td>c</td>
<td>g</td>
<td>g</td>
</tr>
<tr>
<td>Complement Arguments</td>
<td>I</td>
<td>c</td>
<td>g</td>
<td>g</td>
</tr>
<tr>
<td>LG(n)</td>
<td>A</td>
<td>g</td>
<td>g</td>
<td>g</td>
</tr>
<tr>
<td>Permute Arguments</td>
<td>P</td>
<td>g</td>
<td>g</td>
<td>g</td>
</tr>
</tbody>
</table>

Notation:

A = Nonsingular Matrix from LG(n)
P = Permutation Matrix
I = Identity Matrix
a, c = Arbitrary Vector in \(Z^n\)
d = Arbitrary Constant (0 or 1) in Z
\(\mathcal{g}\) = Zero Vector

input variables by adding a constant vector to them. The product of this group with the group of input variable permutations produces the group whose equivalence classes induced on \(\mathcal{F}\) are called "symmetry types" by Polya (1940), Slepian (1967), Singer (1952), and Ashenhurst (1952). When the linear group on \(X\) is combined with argument complementations or vector addition on \(X\), we obtain the full affine group \(\text{AG}(n)\) on the domain of switching functions (Nechiporuk, 1958). Harrison (1963) counted equivalence classes under this group, which has also been studied in the context of finite automata (Gill, 1966; Daykin, 1963).

The third column of Fig. 6 includes those transformation groups which are direct products of domain transformations with range transformations. As previously stated, only two nonsingular range transformations are possible when the function has a single output; either the identity or the functional complementation. The group generated by functional complementation on \(Z\) plus variable permutations and complementations on \(X\) defines a set of
equivalence classes called genera (Ninomiya, 1961). The group $\text{AG}(n) \times \text{AG}(1)$ is the direct product of the affine groups on $X$ and on $Z$, and Harrison (1963) enumerated the number of equivalence classes under this group in 1962. Harrison used a generalization of Polya’s counting theorem (DeBruijn, 1959). This theorem is applicable to direct products of transformation groups on the domain and range of arbitrary mappings (see Chapter IV).

The groups in the right-hand column of Fig. 6 are no longer direct products of domain and range transformation groups, because they involve the addition of linear functions to the output of an arbitrary logic function. Consequently, DeBruijn’s theorem does not apply. Ninomiya (1958) was the first to study such transformation groups, the smallest of which translates the function range in an argument-dependent way by adding linear functions (modulo two sums of arguments). When function complementation, argument complementation, and argument permutations are combined with range translation by linear functions, the result is a subgroup of RAG which partitions functions into equivalence classes that Ninomiya called “families”. Stone and Jackson (1969) discussed properties of certain graphs which interrelate these families; they applied the rather broad term “affine equivalence” to this family relation which is a subgroup of RAG.

The composition of the direct product group $\text{AG}(n) \times \text{AG}(1)$ with addition of linear functions produces the restricted affine group shown on the upper right corner of Fig. 6. This group was first defined by Ninomiya (1958).

From the previous section, $f(x)$ is represented in terms of $g(y)$ and affine mappings on $X$ and $Z$ as follows:

$$f(x) = g(xA \oplus c) \oplus xa^t \oplus d \pmod{2}$$  \hspace{1cm} (107)

4. Groups with Invariant Subspace Restrictions

The preceding paragraph justified the definition of RAG from physical considerations (the need to avoid feedback of a function output back to its input). This section defines a general class of linear and affine subgroups which includes RAG as a special case.

Let $Z^n = Z^{n_1} + Z^{n_2} + \cdots + Z^{n_r}$ be the direct sum of $r$ subspaces $Z^{n_i}$ and consider the subset of all similarly partitioned matrices in $\text{LG}(n)$ with the special quasi-triangular form

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1r} \\ 0 & A_{22} & \cdots & A_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A_{rr} \end{bmatrix}$$  \hspace{1cm} (108)
with \( r \) nonsingular matrices \( A_{ii} \) of dimension \( n_i \) along the main diagonal \( n = n_1 + \cdots + n_r \).

**DEFINITION.** The Restricted Linear Group RLG \((n_1/n_2/\cdots/n_r)\) is the subgroup of LG\((n)\) which consists of all quasitriangular matrices of the above form.

The motivation for the term *restricted* linear group is the following fundamental property of RLG: Let \( s_k = n_1 + n_2 + \cdots + n_k \) for \( 1 \leq k \leq r \). Then, for each \( k \), the set of all \( n \)-tuples whose first \( s_k \) components are zero is a subspace of \( \mathbb{Z}^n \) and is closed under the group RLG \((n_1/n_2/\cdots/n_r)\); i.e., it is an invariant subspace under this RLG. Thus each rectangular zero submatrix restriction that can be defined for RLG matrices produces a corresponding invariant subspace restriction.

**DEFINITION.** Let \( n_1 + n_2 + \cdots + n_r \) be any partition of the integer \( n \). The Restricted Affine Group RAG\((n_1/n_2/\cdots/n_r)\) is that subgroup of AG\((n)\) formed by combining RLG\((n_1/n_2/\cdots/n_r)\) with the additive group of all translations \( x \rightarrow x + c \) on \( \mathbb{Z}^n \).

In our applications, \( r = 2 \), \( n_2 = 1 \), and \( n_1 = n \) will be supplied by the context. Therefore, we will drop the indices from RAG. In the general case of a switching function with \( m \) binary outputs (Lechner, 1963), the notation RAG\((n/m)\) is used to identify \( m \) and \( n \) explicitly. The quantities \( a' \) and \( d \) expand to \( m \) columns, and the unit element in the lower right corner of \( T \) becomes a nonsingular \( m \times m \) matrix.

This zero restriction on submatrices of elements of RAG leads to considerable theoretical difficulty; in fact, no algorithm known to us can reduce an arbitrary member of the restricted affine group to rational canonical form if the number \( m \) of output variables is greater than one. The problem here is that inner automorphisms of RAG (to be admissible as similarity transformations which can produce canonical forms) must also obey this zero submatrix restriction; otherwise, they are not physically realizable as feedback-free encoding transformations. Fortunately, rational canonical forms of restricted affine transformations have been completely characterized in the special case of a single output function in which \( m = 1 \) and \( a \) is a row vector, not a matrix of \( m > 1 \) rows. This characterization is constructive in the sense that it defines an algorithm by which a single prototype encoding transformation can be reduced to rational canonical form (as far as it is possible to do so) by similarity transformations that are inner automorphisms of the group RAG.
This construction was applied to count equivalence classes of prototype transformations, but heuristic techniques based on harmonic analysis may actually be more effective in actual synthesis of the associated encoding transformations.

5. An Isomorphism between $\text{RLG}(1/n/1)$ and $\text{RAG}(n/1)$

A natural isomorphism exists between $\text{RAG}(n/1)$ operating on $\mathbb{Z}^{n+1}$ and $\text{RLG}(1/n/1)$ acting on the set of all vectors $(1; x; z)$ with $x \in \mathbb{Z}^n$ and $z \in \mathbb{Z}$. This set is a coset of the invariant subspace $(0; X; Z)$ of $\mathbb{Z}^{n+2}$. The correspondence preserves the identity of each component $A$, $a$, $c$, and $d$ of the matrix form of $\text{RAG}$ (See Eq. (103)), and it permutes the $x$ and $z$ components of the coset $(1; x; z)$ in the same manner as the corresponding element of $\text{RAG}$.

$$T = \begin{bmatrix}
1 & c & d \\
0^t & A & a^t \\
0 & 0 & 1
\end{bmatrix} = T_{A, a, c, d} \quad (109)$$

To verify that this matrix has the desired effect on $(1; X; Z)$, premultiply it by $(1; x; z)$ and observe the desired result.

$$(1; x; z)T = (1; xA \oplus c; z \oplus xa^t \oplus d) \quad (110)$$

**Theorem 4.1.** The subgroup $\text{RLG}(1/n/1)$ of $\text{LG}(n + 2)$ is isomorphic to the subgroup $\text{RAG}(n/1)$ of $\text{AG}(n + 1)$.

**Proof:** The correspondence defined by inserting components $A$, $a$, $c$, $d$ of Eq. (106) into the matrix $T_{A, a, c, d}$ is obviously one to one. Preservation of the group operation can be verified directly by matrix multiplication which is left as an exercise.

This isomorphism is an important step in the reduction of $\text{RAG}$ elements to rational canonical form (a necessary preliminary to enumeration of equivalence classes). Generalization to groups with more general partitions of $X$ and (for $m > 1$) of $\mathbb{Z}^m$ is straightforward.

6. The Weight-Transforming Property of $\text{RAG}$

Transformation groups acting on $X$ alone merely permute points. In contrast, a general element of $\text{RAG}$ does not preserve the weight of $f$ (size or cardinality of $f^{-1}(1)$) unless $a$ and $d$ are zero. The following theorem (really
a corollary of the fundamental invariance theorem of Section II,C) defines a specific set of different possible weights which characterizes the members of a given prototype class. This set of weights is defined explicitly by the difference between $2^{n-1}$ and each member of the spectrum or set of integer magnitudes which is included in the range of the Fourier transform $f_N$. 

**Theorem 4.2.** The prototype class which contains $f$ contains at least one function $g$ whose weight (size of $g^{-1}(1)$) is $m$ iff $m$ is a member of the set \( \{2^{n-1} \pm f_N(a) : a \in \mathbb{Z}^n \} \).

**Proof:** By the transform construction of Section II,A,4, \( |f^{-1}(1)| = f^*(0) = 2^{n-1} - f_N(0) \). If $g = f$, then $g = f \oplus 1$ and $g^*(0) = 2^n - f^*(0) = 2^{n-1} + f_N(0)$. Now let $g = f \oplus xa^t \oplus d$. By the preceding theorem, $g^*(0) = (-1)^d f_N^*(a) = 2^{n-1} - g^*(0) = 2^{n-1} - |g^{-1}(1)|$.

Therefore, $g^{-1}(1) = 2^{n-1} - g^*(0) = 2^{n-1} - (-1)^d f_N^*(a) = m$ iff $m = 2^{n-1} \pm f_N^*(a)$ for some $a \in \mathbb{Z}^n$. The value of $d$ (which is arbitrary) determines the sign in this equation.

Using this theorem, it is a simple matter to compute the set of possible weights of functions which are derivable from a given function $f$ by adding a linear function $xa^t$. The subset of vectors $a$ for which $f \oplus xa^t$ has this weight is also directly identifiable from $f_N$.

**B. ENUMERATION OF PROTOTYPE EQUIVALENCE CLASSES**

The problem of simply counting equivalence classes of $n$-input, single-output functions under RAG is complicated by the fact that RAG is not merely a domain transformation group like the symmetry group studied by Slepian and many others. DeBruijn's generalization of Polya's theorem was used successfully by Harrison to count classes under direct products of domain and range transformation groups (see Chapter IV). However, RAG is not a direct product group, so DeBruijn's theorem does not apply. Yet not only the counting problem (how many classes are there?), but also the recognition problem (to which class does a given function belong?) have been solved completely for RAG for $n \leq 4$ (Ninomiya, 1958). Unfortunately, Ninomiya's semiexhaustive technique is difficult to extend to five variables.

The case $n = 5$ was enumerated by a new technique for counting
functional equivalence classes under RAG (Lechner, 1963a,b). The main result was that 48 prototype classes are now known to exist for \( n = 5 \). Unfortunately, the counting procedure does not explicitly define representatives for equivalence classes. Therefore, another approach to identification of class representatives is described in Section IV,C, followed by a tabulation of 46 out of the 48 prototype equivalence classes for \( n = 5 \).

1. Application of Polya’s Theorem

The counting technique employed by Lechner (1963) appears to involve less restrictive conditions than DeBruijn’s theorem. A function \( f \) from \( X \) to \( Z \) is represented uniquely by its truth table, which is the set of all ordered pairs \((x, z)\) such that \( z = f(x) \). Now \( f \) can also be regarded as an abstract relation (i.e., as a subset of \( X + Z \) which is called the “graph” of \( f \)). An alternate representation of \( f(x) \) is by the characteristic function of its graph, i.e., a binary function \( C_f(x, z) \) such that \( C_f(x, z) = 1 \) if \( z = f(x) \), and zero otherwise. These two representations for \( f \) are illustrated for a 3-input, 2-output example in Table IV.

Since RAG is a subgroup of the general affine group on \( X + Z \), it permutes points in the domain of \( C_f \). It is easy to show that the set of all single-valued

<table>
<thead>
<tr>
<th>A. Truth Table</th>
<th>B. Graph ( C_f(x, z) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x \in X )</td>
<td>( z \in Z^2 )</td>
</tr>
<tr>
<td>( (x_1, x_2, x_3) )</td>
<td>( (f_1(x), f_2(x)) )</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>X:</td>
<td>( z^2: f_1(x) )</td>
</tr>
<tr>
<td>X:</td>
<td>( f_2(x) )</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 1 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 1 0 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 0 1 1</td>
</tr>
</tbody>
</table>
relations (i.e., functions) is closed under RAG. Therefore, the following form of Polya's theorem can be applied to enumerate equivalence classes, provided that the required parameters can be evaluated:

**THEOREM 4.3** (Polya, 1937). The number $N_{F/G}$ of equivalence classes into which the set $F$ of all subsets of a finite set $S$ is partitioned under a finite group $G$ of permutations on $S$ is defined by:

$$N_{F/G} = \sum_{C_i \in G} [N_{F/C_i}/N_{C_i}]$$  \hspace{1cm} (111)

where the sum is over all equivalence classes $C_i$ of elements in $G$ under its group of inner automorphisms, $N_{F/C_i} = 2^{N_{S/C_i}}$, is the number of subsets of $S$ left invariant by any one element $T$ of class $C_i$, $N_{S/C_i}$ is the number of transitive subsets or cycles into which $S$ is partitioned by repeated application of $T$, and $N_{C_i}$ is the number of inner automorphisms of $G$ which leave $T$ invariant.

This theorem is derivable from Harrison's statement of a theorem of Frobenius (Theorem 2.1) in Chapter IV of this book. The symbols $\alpha$, $S$, and $I(\alpha)$ of Harrison correspond directly to $T$, $F$, and $N_{F/C_i}$ of our Theorem 4.3. Harrison's sum over all group elements $\alpha$ can be replaced by a sum over conjugate classes $C_i$ of group elements, because all group elements $\alpha$ in the same class $C_i$ leave invariant the same number $I(\alpha) = N_{F/C_i}$ of subsets of $S$. The $N_{C_i}$, $T$-invariant inner automorphisms of $G$ themselves form a subgroup (see Harrison's Fact 2.3 in Chapter IV) with $n_i$ cosets in $G$. Therefore, $g = n_iN_{C_i}$, and Harrison's formula becomes:

$$(1/g) \sum_{\alpha} I(\alpha) = (1/g) \sum_{i} N_{F/C_i} n_i = \sum_{i} N_{F/C_i}/N_{C_i}$$  \hspace{1cm} (112)

which is Theorem 4.3.

In our application, $S = X + Z$, $G = RAG$, and $F$ is the set of characteristic functions corresponding to truth tables. This form of Polya's theorem depends on the fact that every element of an equivalence class $C_i$ of RAG under its group of inner automorphisms leaves invariant the same number of single-valued functions (Theorem 6 of Lechner, 1963). To apply Polya's theorem, it is necessary to determine the cycle structure of a typical element of every class $C_i$ and the number $N_{C_i}$ of $T$-invariant inner automorphisms of $G$. Methods for constructively defining these parameters for the group RAG were evolved by Lechner (1963). However, for the sake of brevity, details will be omitted herein.

Fortunately, the number of classes is generally far smaller than the number of elements of $G$. The parameter $N_{F/C_i}$ is zero for all but a small fraction of the $C_i$. This helps to reduce the computations.
2. Summary of Results

Table V gives a lower bound for the order of RAG, the total number of classes \( C_i \) in RAG, and finally the number of classes whose members leave single-valued functions invariant for \( 2 \leq n \leq 5 \).

Table VI summarizes the results obtained for the number of classes into which RAG partitions \( F \), together with previously published data on other groups for comparison.

**TABLE V**
Structure of Classes in RAG

<table>
<thead>
<tr>
<th>( n = )</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order (RAG)</td>
<td>192</td>
<td>21,504</td>
<td>10,321,924</td>
<td>( 20 \cdot 10^9 )</td>
</tr>
<tr>
<td>Number of Classes ( C_i )</td>
<td>13</td>
<td>28</td>
<td>62</td>
<td>124</td>
</tr>
<tr>
<td>Classes with ( N_F/C_i \neq 0 )</td>
<td>7</td>
<td>13</td>
<td>32</td>
<td>62</td>
</tr>
</tbody>
</table>

**TABLE VI**
Comparison of Equivalence Class Counts

<table>
<thead>
<tr>
<th>Group</th>
<th>( n = )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAG</td>
<td>1 2 3 4 5</td>
<td>&gt;69,000</td>
<td>&gt;10^{19}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AG(n) x AG(1)</td>
<td>2 3 6 8</td>
<td>206</td>
<td>7,888,269</td>
<td>&gt;10^{21}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AG(n)</td>
<td>3 5 10 32</td>
<td>382</td>
<td>15,768,919</td>
<td>&gt;10^{21}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LG(n)</td>
<td>4 8 20 92</td>
<td>2,744</td>
<td>950,998,216</td>
<td>&gt;10^{24}</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( n )-cube symmetries</td>
<td>3 6 22 402</td>
<td>1,228,158</td>
<td>&gt;10^{14}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table VII lists representatives for the eight prototype classes when \( n = 4 \), as chosen by Ninomiya (1958). Ninomiya’s table of symmetry types and their class memberships has been reprinted by Harrison (1965). All but one prototype is linearly separable. The numbers of families, domain classes, genera, and functions in each prototype class are also tabulated, along with the Fourier coefficient magnitudes (spectrum) which identifies prototype class membership for \( n \leq 4 \) (see Fig. 2).

The Fourier spectrum of a function \( f(x) \) with range \( (0, 1) \) is defined as the unordered set of positive integers which are the absolute values of the coefficients of the Fourier transform \( f_N^* \) (i.e., the transform of the (real-valued function \( \frac{1}{2} - f(x) \)) defined in Section III. The notation \( k^n \) in the last column
TABLE VII
Decomposition into Equivalence Classes

<table>
<thead>
<tr>
<th>Prototype Index</th>
<th>Representative Function</th>
<th>Threshold Gate Realization</th>
<th>Number of Families</th>
<th>Domain Classes</th>
<th>Genera</th>
<th>Functions</th>
<th>Fourier Coefficient Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>$y_1y_2y_3y_4$</td>
<td>$y_1y_2y_3y_4 = 4$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>$y_1y_2y_3$</td>
<td>$y_1y_2y_3y_4 = 3$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>512</td>
</tr>
<tr>
<td>6</td>
<td>$y_1y_2y_3$</td>
<td>$y_1y_2y_3$</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>30</td>
<td>2840</td>
</tr>
<tr>
<td>5</td>
<td>$y_1y_2y_3 + y_4$</td>
<td>$y_1y_2y_3y_4 = 5$</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>52</td>
<td>17,920</td>
</tr>
<tr>
<td>4: A</td>
<td>$y_1^2y_2$</td>
<td>$y_1y_2 = 2$</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>19</td>
<td>1120</td>
</tr>
<tr>
<td>4: B</td>
<td>$y_1y_2y_3y_4$</td>
<td>$y_1y_2y_3 + y_4 = 4$</td>
<td>1</td>
<td>13</td>
<td>3</td>
<td>76</td>
<td>26,864</td>
</tr>
<tr>
<td>3</td>
<td>$y_1y_2y_3y_4$</td>
<td>$y_1y_2y_3y_4 = 3$</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>33</td>
<td>14,336</td>
</tr>
<tr>
<td></td>
<td>$\sqrt{y_2y_3y_4}$</td>
<td></td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>896</td>
</tr>
<tr>
<td></td>
<td>$\sqrt{y_2y_3y_4 \sqrt{y_2y_3y_4}}$</td>
<td>Sum of Index 3 and Index 7 Prototypes</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>896</td>
</tr>
</tbody>
</table>

* Under the affine group on the domain plus functional complementation.
* Index 7 prototype with complemented variables as inputs.

of Table VII means there are $n$ coefficients equal to $\pm k$ among the transform coefficients. For example, $8^30^{15}$ means 15 of the coefficients are 0-valued, and one of them is equal to $\pm 8$.

3. The Three-Argument Case

Ninomiya (1961) studied the correspondence between symmetry types and prototypes in detail for functions of $n \leq 4$ arguments. For $n = 3$, the transformations can be determined by inspection. Particular solutions for the encoding transformation that converts symmetry types into prototypes are shown in Table VIII. The "+" sign represents Boolean INCLUSIVE OR addition in this table.

For the 3-input, 1-output situation, only 14 different mappings are required. Each of these converts a generic class into a prototype class; each genus is a union of symmetry types under the equivalence relation induced by functional complementation. Each generic class is a different row of Table VIII. For example, generic class 3 is in prototype class 2 and contains 12 different functions. The 12 different functions in this class may be obtained
<table>
<thead>
<tr>
<th>Prototype Class</th>
<th>Generic Class</th>
<th>Functions Per Class</th>
<th>Standard Sum (Minterms in $f(x)$)</th>
<th>Normal Form</th>
<th>Number of Inverters Plus AND/OR Gates</th>
<th>Linearly Encoded Form</th>
<th>Number of AND Gates Plus Exclusive OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>6</td>
<td>4,5,6,7</td>
<td>$x_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>6</td>
<td>2,3,4,5</td>
<td>$x_1 \overline{x}_2 \oplus \overline{x}_1 x_2$</td>
<td>5</td>
<td>$x_1 \oplus x_2$</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>2</td>
<td>1,2,4,7</td>
<td>$x_1 x_2 + x_1 \overline{x}_2 + x_2 \overline{x}_3 + \overline{x}_1 x_2 \overline{x}_3$</td>
<td>7</td>
<td>$x_1 \oplus x_2 \oplus x_3$</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>8</td>
<td>7</td>
<td>$x_1 x_2 x_3$</td>
<td>1</td>
<td>$x_1 x_2 x_3$</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>24</td>
<td>5,6,7</td>
<td>$x_1 x_2 + x_1 x_3$</td>
<td>3</td>
<td>$x_1 x_2 x_3 \oplus x_1$</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>24</td>
<td>1,4,7</td>
<td>$x_1 x_2 + x_1 \overline{x}_2 + \overline{x}_1 x_2 x_3$</td>
<td>5</td>
<td>$x_1 x_2 x_3 \oplus x_1 \oplus x_2 \oplus 1$</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>8</td>
<td>3,5,6</td>
<td>$x_1 x_2 x_3 + x_1 x_2 \overline{x}_3 + x_1 \overline{x}_2 x_3$</td>
<td>7</td>
<td>$x_1 x_2 x_3 \oplus x_1 \oplus x_2 \oplus \overline{x}_3 \oplus 1$</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>12</td>
<td>6,7</td>
<td>$x_1 x_2$</td>
<td>1</td>
<td>$x_1 x_2$</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>24</td>
<td>0,5,6,7</td>
<td>$x_1 x_2 + x_1 x_3 + x_1 \overline{x}_2 x_3$</td>
<td>7</td>
<td>$x_1 x_2 x_3 \oplus x_1$</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>12</td>
<td>4,7</td>
<td>$x_1 x_2 x_3 + x_1 \overline{x}_2 x_3$</td>
<td>5</td>
<td>$x_1 x_2 x_3 \oplus x_2 \oplus 1$</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>24</td>
<td>3,4,6,7</td>
<td>$x_2 x_3 + x_1 x_3$</td>
<td>4</td>
<td>$(x_1 \oplus x_2) \oplus x_3 \oplus x_1$</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
<td>0,7</td>
<td>$x_1 x_2 x_3 + x_2 \overline{x}_3 + \overline{x}_1 x_2 x_3$</td>
<td>6</td>
<td>$(x_1 \oplus x_2 \oplus 1)$</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>8</td>
<td>3,5,6,7</td>
<td>$x_1 x_2 + x_2 x_3 + \overline{x}_1 x_3$</td>
<td>4</td>
<td>$(x_1 \oplus x_2 \oplus 1)$</td>
<td>9</td>
</tr>
</tbody>
</table>

TOTAL: 256  
AVERAGE COST: 2.7  
2.1
by complementing the function and/or permuting and/or complementing input variables in all possible ways.

Some of the linearly encoded "prototype" functions in Table VIII require one 2- or 3-input AND gate in addition to the EXCLUSIVE OR's, but no function requires more than one. Based on the duality theorem \( (\bar{x} \lor \bar{y} = xy \oplus 1) \), logical sum rather than logical product prototype forms appear to be more economical (e.g., generic class 7 becomes \( (x_1 \lor x_2 \lor x_3) \oplus x_4 \oplus x_2 \)). However, this depends on the generic class representative, since variable and functional complementation are used to map functions within each generic class.

The average number of gates and/or inverters is also shown on Table V-8 for each generic class. The products of these numbers with the number of functions per class, summed over all 14 classes and divided by 256 gives an average cost (for randomly chosen functions) of 2.7 gates or inverters for the normal form realization and 2.1 gates or inverters for the linearly encoded form. (However, inversions within each generic class were not considered in the average.) Inverters were equated with gates for the normal form realization, because inversion is accomplished by EXCLUSIVE OR gates in the linearly coded version. EXCLUSIVE OR gates were equated with AND/OR gates because circuits of equivalent complexity are expected to result for batch fabricated (LSI) arrays of EXCLUSIVE OR circuits (fanout restrictions compensate for the inherently greater complexity of EXCLUSIVE OR gates).

4. Number of Classes versus Function Weight

Since affine domain transformations merely permute points of \( X \), it is of interest to identify the number of equivalence classes of functions under \( AG(n) \) on \( X \), separately for each weight \( m = |f^{-1}(1)| \) (size or cardinality of the subset \( f^{-1}(1) \)). This has been done by Nechiporuk (1958) for \( n \leq 5 \) and all \( m \). A similar computation was carried out by Slepian (1960) for the linear group \( LG(n) \). Slepian's data (for group codes) was modified by Lechner (1967) for the context of switching functions. Table IX compares the results of Nechiporuk and Slepian for \( n \leq 5 \).

Slepian's data actually extended to \( n \leq 9 \) and \( m \leq 19 \). Table X from Lechner (1967) presents this data in its entirety.

Since \( RLG(1/n-1) \) or \( RAG(n-1) \) is a subgroup of both \( LG(n+2) \) and \( AG(n+1) \), the data in Tables IX and X might help to establish bounds on the number of equivalence classes under \( RAG(n-1) \). By the theorem on weight-transformation in Section IV,A,6, equivalence classes under \( RAG(n-1) \) are actually unions of a small number of equivalence classes under \( AG(n) \) of different weights. The latter are directly determined by the number of distinct coefficient magnitudes in \( f_N^* \).
<table>
<thead>
<tr>
<th>(m)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5&lt;sup&gt;b&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>3</td>
<td>11</td>
<td>9</td>
<td>55</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>3</td>
<td>9</td>
<td>15</td>
<td>84</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>2</td>
<td>7</td>
<td>16</td>
<td>117</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
<td>23</td>
<td>158</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td>1</td>
<td>3</td>
<td>24</td>
<td>204</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>30</td>
<td>242</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>30</td>
<td>274</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>38</td>
</tr>
<tr>
<td>Totals:</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>10</td>
<td>32</td>
</tr>
</tbody>
</table>

* Data on \(N_{A_{m,n}}\) came from Nechiporuk (1958); data on \(N_{m,n}\) came from Table X.

<sup>b</sup> For \(n = 5\), the identity \(N_{A_{m,n}} = N_{A_{2n-1,m,n}}\) is used; similarly for \(N_{m,n}\)
TABLE X
The Number of Equivalence Classes of Switching Functions of \( n \) Arguments and Weight \( m \) or \( 2^n - m \).*

<table>
<thead>
<tr>
<th>( m )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>9</td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>11</td>
<td>23</td>
<td>29</td>
<td>30</td>
<td>30</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>12</td>
<td>35</td>
<td>54</td>
<td>61</td>
<td>62</td>
<td>62</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>55</td>
<td>107</td>
<td>135</td>
<td>143</td>
<td>144</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>84</td>
<td>227</td>
<td>329</td>
<td>368</td>
<td>377</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>7</td>
<td>117</td>
<td>495</td>
<td>887</td>
<td>1,070</td>
<td>1,122</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>158</td>
<td>1,131</td>
<td>2,728</td>
<td>3,706</td>
<td>4,010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>204</td>
<td>2,698</td>
<td>9,615</td>
<td>16,337</td>
<td>17,748</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>242</td>
<td>6,615</td>
<td>39,039</td>
<td>85,028</td>
<td>101,210</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>274</td>
<td>16,488</td>
<td>180,773</td>
<td>538,627</td>
<td>770,334</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>290</td>
<td>41,016</td>
<td>923,922</td>
<td>4,430,674</td>
<td>7,920,75</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>274</td>
<td>99,965</td>
<td>1,19623</td>
<td>42,9759</td>
<td>105,524</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>242</td>
<td>235,547</td>
<td>27,3633</td>
<td>458,395</td>
<td>1,811,13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>204</td>
<td>530,998</td>
<td>147,945</td>
<td>5103,84</td>
<td>35432,4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Integer larger than \( 10^6 \) are rounded off to six significant figures; entries containing decimal points should be multiples by \( 10^6 \).

C. EXPLICIT DEFINITION OF 5-VARIABLE PROTOTYPES

In order to use the prototype equivalence relation for synthesis, it is convenient to have an explicit tabulation of prototype class representatives and a simple means of identifying the class to which an arbitrary function belongs. For \( n = 5 \), there are only 48 classes (Lechner, 1963a,b); therefore, an exhaustive tabulation was attempted, and the following results are reproduced from Lechner (1968). For \( n = 6 \), exhaustive tabulation of the (more than 69,000) classes might also be feasible, although its utility is doubtful. For \( n \geq 7 \), there
are too many classes to tabulate. However, other synthesis techniques which
require neither tabulation nor recognition will be proposed in Section V. The
catalog of prototypes in Table XI of this section verifies the conjecture of
Ninomiya (1958) that the spectrum of a function of five or more inputs does
not uniquely determine its prototype class. Using this table, we can determine
the prototype class of an arbitrary five-input function to within a small
ambiguity.

A Monte Carlo technique was used to identify 40 different spectra; an
argument based on invariance of level set rank under RLG(1/\pi/1) showed that
six of these spectra are associated with at least two distinct prototype classes
(“split” classes). Two of the 48 prototype classes are still not identified. These
two classes may have spectra that are distinct from the 40 varieties already
found, or they may result from additional class-splitting of one of the six
identified split classes, or breakup of one of the 34 other classes.

1. Monte Carlo Approach

The Monte Carlo analysis used a pseudorandom binary sequence genera-
tor to produce a total of 132,000 distinct 5-input functions. The truth tables
for most of these functions were defined by adjacent but nonoverlapping
32-bit segments of the sequence generator’s output. The Fourier transforms of
all these functions were computed and stored on tape. This tape was then
re-sorted and scanned to produce a list of the distinct spectra and a count of
the number of times each spectrum appeared among the entire sample of
functions.

The first computer run generated 130,000 different functions starting from
a randomly chosen initial state of the sequence generator and detected 36
distinct spectra. Next, we made a shorter run (2000 functions) starting from an
initial condition that was specially chosen for its rarity: a zero value was
assigned to 31 of its 32 bit positions. Four new spectra were discovered, but
the previously discovered spectra reappeared with consistently greater
frequency. The apparent degeneracy of this special initialization process into
pseudorandom behavior discouraged us from further attempts at exhaustive
sampling.

The initial sample (130,000 functions) is an extremely small fraction of the
total number of 5-input functions (approximately four billion). The percents-
tage with which functions having the same spectrum reappeared in this
sample are also tabulated. (An asterisk means the spectrum appeared only in
the smaller sample of 2000 functions.) These frequencies vary widely and
give some idea of the relative sizes of different prototype equivalence classes
(or pairs of classes if they have the same spectrum).
We do not know the extent to which a choice for the sequence generator's initial conditions affects the delay before the first appearance of a function with a given spectrum. Neither do we know the sequence length necessary to identify all nontrivial spectra. From an engineering standpoint this is not too important since the functions found in practical applications also depart from randomness to an unknown, but probably more significant, extent. For example, the number of 1's and 0's in a randomly chosen truth table has a binomial distribution, but real-life examples from logic design seem to be biased toward a much lower or higher number of 1's.

2. Detection of Split Classes

The first split class was detected accidentally while attempting to map a class representative for each spectrum into a unique prototype. To produce a unique prototype, we followed Ninomiya's lead and tried to derive an encoding transformation which permuted the transform coefficients into a maximal lexicographic ordering within the transform vector $f_N^*$. Standard sums $f^{-1}(1)$ for the particular prototype class representatives that resulted from this attempt are listed in Table XI at the end of this section.

In several cases, we found it impossible to map one function into another with the same spectrum. A constructive proof was then derived to show that such a transformation could not exist within the restricted affine group. Thus, the existence of six split classes was demonstrated. These split classes are identified in the table as pairs $A, B$ within the same type number designation. The Fourier transform coefficient vectors are listed for each split class pair in Table XII below. Proof that functions with these spectra actually split into two equivalence classes under RAG is based on the invariance of level set ranks under the group $RLG(1/n/1)$. It is not known whether the spectrum (size of each level set) plus the ranks of each level set (as a subset of $Z^*$) is always sufficient to resolve the ambiguity.

3. Explanation of Results

Each row of Table XI contains (from left to right) a prototype class number, the spectrum of all members of that class, the approximate size of that class, the standard sum of the prototype function chosen as the class representative, and a remarks column.

Class numbers have been assigned in descending lexicographic order of the spectrum of the class. The six cases in which two prototype classes have the same spectrum have been labeled $A$ and $B$ and contain two standard sums. Table XII also identifies the complete Fourier transform coefficient vector $f_N^* = (\frac{1}{2} - f)^*$ for both prototype representatives in each split class.
<table>
<thead>
<tr>
<th>Prototype Class No.</th>
<th>Spectrum</th>
<th>Size of Class</th>
<th>Standard Sum ([\frac{1}{16} - 1])</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16(^0) (31)</td>
<td>*</td>
<td>(no terms)</td>
<td>Degenerate class (n=0)</td>
</tr>
<tr>
<td>2</td>
<td>15(^1) (21)</td>
<td>*</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>14(^2) 15(16)</td>
<td>*</td>
<td>30, 31</td>
<td>Degenerate class (n=4)</td>
</tr>
<tr>
<td>4</td>
<td>13(^3) (24)</td>
<td>0.01</td>
<td>29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>12(^4) (24)</td>
<td>&lt; 0.01</td>
<td>28, 29, 30, 31</td>
<td>Degenerate class (n=3)</td>
</tr>
<tr>
<td>6</td>
<td>11(^5) (16) (12)</td>
<td>0.05</td>
<td>27, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10(^6) (24) (24)</td>
<td>0.04</td>
<td>27, 28, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9(^7) (10) (20)</td>
<td>0.24</td>
<td>23, 27, 29, 30, 31</td>
<td>Degenerate class (n=4)</td>
</tr>
<tr>
<td>9</td>
<td>8(^8) (22) (16)</td>
<td>0.02</td>
<td>26, 27, 28, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>7(^9) (16) (12) (12)</td>
<td>0.60</td>
<td>23, 27, 28, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>6(^10) (20)</td>
<td>0.05</td>
<td>16, 23, 27, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>5(^11) (10) (10)</td>
<td>0.64</td>
<td>15, 23, 27, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>4(^12) (28)</td>
<td>&lt; 0.01</td>
<td>25 thru 31</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>3(^13) (24) (28)</td>
<td>0.57</td>
<td>22, 28 thru 31</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>2(^14) (13) (18)</td>
<td>0.40</td>
<td>19, 23, 27, 28, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1(^15) (19)</td>
<td>3.3</td>
<td>15, 23, 27, 28, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0(^16) (15)</td>
<td>0.66</td>
<td>15, 16, 23, 27, 28, 30, 31</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>3(^17) (19)</td>
<td>*</td>
<td>24 thru 31</td>
<td>Degenerate class (n&gt;7)</td>
</tr>
<tr>
<td>19</td>
<td>2(^18) (22)</td>
<td>0.07</td>
<td>23, 25 thru 31</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1(^19) (22)</td>
<td>0.03</td>
<td>22, 23, 26, 27, 28, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>0(^20) (10)</td>
<td>0.45</td>
<td>21, 23, 26, 27, 28, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>3(^21) (14) (11)</td>
<td>4.9</td>
<td>15, 23, 27, 28, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>2(^22) (16) (10)</td>
<td>6.7</td>
<td>15, 19, 25, 27, 28, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>1(^23) (19)</td>
<td>0.41</td>
<td>3, 15, 23, 27, 28, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>0(^24) (20)</td>
<td>5.5</td>
<td>14, 15, 16, 23, 27, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>3(^25) (21)</td>
<td>0.50</td>
<td>15, 23, 29 thru 31</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>2(^26) (22)</td>
<td>0.62</td>
<td>15, 22, 23, 29 thru 31</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>1(^27) (20) (18)</td>
<td>7.4</td>
<td>15, 21, 23, 26 thru 31</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>0(^28) (19)</td>
<td>9.8</td>
<td>7, 15, 23, 26 thru 31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1(^29) (15)</td>
<td>12.3</td>
<td>14, 15, 20, 23, 25, 27, 29, 30, 31</td>
<td>Split class</td>
</tr>
<tr>
<td>30A</td>
<td>1(^30) (15)</td>
<td>14, 15, 20, 23, 25, 27, 29, 30, 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30B</td>
<td>1(^31) (28)</td>
<td>1,15, 23, 28 thru 31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The spectrum of a function is a list of the number of occurrences of each integer magnitude as a coefficient of the transform \(f_N^* = (\frac{1}{16} - f)^*\). The spectrum is invariant for all members of a given class. The notation 16\(^0\) \(31\) for the spectrum of class number 1 means that the coefficient ± 16 appears once and 0 appears 31 times in the Ninomiya transform of the function. The positions and signs of these coefficient magnitudes vary over the functions within each class. This is illustrated by the split classes (e.g., 30A and B) for
### TABLE XI (Continued)

<table>
<thead>
<tr>
<th>Prototype Class No.</th>
<th>Spectrum</th>
<th>Size of Class</th>
<th>Standard Sum ( {1/f_1 = 1} )</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>(6^2,10,16)</td>
<td>0.02</td>
<td>14, 15, 22, 23, 28 thru 31</td>
<td>Degenerate class (n=4)</td>
</tr>
<tr>
<td>32</td>
<td>(6^4,4,12,12)</td>
<td>2.7</td>
<td>13, 15, 22, 23, 26 thru 31</td>
<td>Split class</td>
</tr>
<tr>
<td>32A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>(6^6,28)</td>
<td>0.77</td>
<td>7, 15, 23, 26 thru 31</td>
<td>Split class</td>
</tr>
<tr>
<td>33A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33B</td>
<td>(6^4,6,10,10)</td>
<td>10.0</td>
<td>14, 15, 21, 23, 24, 27 thru 31</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>(6^2,8,16,8)</td>
<td>15.7</td>
<td>7, 15, 23, 26 thru 31</td>
<td>Split class</td>
</tr>
<tr>
<td>35</td>
<td>(6^2,10,15,6)</td>
<td>3.9</td>
<td>13, 14, 16, 22, 23, 25, 27, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>(6^3,10,16)</td>
<td>10.5</td>
<td>1, 13, 14, 22, 23, 26 thru 31</td>
<td>Split class</td>
</tr>
<tr>
<td>37</td>
<td>(6^3,10,16)</td>
<td>10.5</td>
<td>14, 15, 22, 23, 26 thru 31</td>
<td>Split class</td>
</tr>
<tr>
<td>38</td>
<td>(6^3,10,16)</td>
<td>10.5</td>
<td>13, 14, 15, 22, 23, 25, 27, 29, 30, 31</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>(6^3,10,16)</td>
<td>10.5</td>
<td>1, 13, 14, 22, 23, 26 thru 31</td>
<td>Split class</td>
</tr>
<tr>
<td>40</td>
<td>(6^3,10,16)</td>
<td>10.5</td>
<td>1, 13, 14, 22, 23, 26 thru 31</td>
<td>Split class</td>
</tr>
</tbody>
</table>

### TABLE XII

Fourier Transform Coefficient Vectors for Representatives of Split Classes \((f_n^*, w(j))\) versus \(j\)

<table>
<thead>
<tr>
<th>Prototype Class No.</th>
<th>Value of (j)</th>
<th>(w = w = \cdots)</th>
<th>(w = w = \cdots)</th>
<th>(w = w = \cdots)</th>
<th>(w = w = \cdots)</th>
<th>(w = w = \cdots)</th>
<th>(w = w = \cdots)</th>
</tr>
</thead>
<tbody>
<tr>
<td>30A</td>
<td>7</td>
<td>5 5 5 3 3</td>
<td>3 3 3 3 3</td>
<td>3 3 3 3 3</td>
<td>3 3 3 3 3</td>
<td>3 3 3 3 3</td>
<td>3 3 3 3 3</td>
</tr>
<tr>
<td>30B</td>
<td>7</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
</tr>
<tr>
<td>32A</td>
<td>6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
</tr>
<tr>
<td>32B</td>
<td>6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
</tr>
<tr>
<td>33A</td>
<td>6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
</tr>
<tr>
<td>33B</td>
<td>6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
</tr>
<tr>
<td>34A</td>
<td>6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
</tr>
<tr>
<td>34B</td>
<td>6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
</tr>
<tr>
<td>35A</td>
<td>6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
</tr>
<tr>
<td>35B</td>
<td>6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
<td>6 6 6 6 6</td>
</tr>
<tr>
<td>36A</td>
<td>5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
</tr>
<tr>
<td>36B</td>
<td>5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
</tr>
<tr>
<td>37A</td>
<td>5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
</tr>
<tr>
<td>37B</td>
<td>5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
</tr>
<tr>
<td>38A</td>
<td>5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
</tr>
<tr>
<td>38B</td>
<td>5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
<td>5 5 5 5 5</td>
</tr>
</tbody>
</table>
which Table XII lists two different complete transform vectors having the same spectrum.

The size of each class is the percentage of occurrences of functions with the given spectrum out of the first pseudorandom sample of 130,000 functions. Four spectra that appeared only in the second sample of 2000 functions are identified by an asterisk in this column.

The last column of Table XII identifies both split classes and degenerate classes, i.e., classes whose prototype representatives are functions of \( n \leq 4 \) variables. There are exactly eight degenerate classes; these correspond to the eight prototype classes of four variables listed in Table VII.

**EXERCISES**

12. In each block of Fig. 5a–c, supply the matrix equation which defines the appropriate component of an element of AG\((n) \times AG(1)\), AG\((n + 1)\), or \( RAG(n/1) \), respectively.

13. Using the cartesian product representation of \( X + Z^m \) as on part B of Table IV (with \( x \) indexing rows and \( y \) indexing columns), demonstrate the proper subgroup inclusion relationships \( RAG(n/m) < S_n \times S_m^{n} < S_{n+m} \) where \( S_n \) is the symmetric group of permutations \( P \) of the \( 2^n \) rows of \( X + Z^m \), and \( S_m^{n} \) is the direct product of \( 2^n \) permutation groups \( Q_x, x \in X \), where \( Q_x \) permutes the elements within row \( x \) of \( X + Z^m \).

14. By postmultiplying \((x, g)\) first by \( T_{A, a, c, d} \) and then by \( T_{B, b, e, f} \), derive the multiplication rule for elements of \( RAG \).

15. Define a partial ordering among the symbols \(|, P, A, 0, a, c, d|\) on Table III which will produce the subgroup ordering on the lattice diagram of Fig. 6.

**V. SYNTHESIS OF ENCODED INPUT LOGIC**

In this section, the prototype equivalence relation is proposed as a basis for actual synthesis. A transformation from the restricted affine group (called an "encoding" transformation) is used to map each desired function \( f \) into a simpler prototype function \( g \) in the same equivalence class. If "don't care" states appear, they are selected so as to yield the simplest prototype or (more generally) to minimize total cost for prototype plus encoding transformation.

The prototype is designed according to conventional methods and imbedded within the encoding transformation to realize the desired function \( f \). The
encoding transformation can be used to simplify either a minimal covering of the function by prime implicants, or the number of threshold elements, if a threshold logic realization is desired. Of course, functions with iterative decompositions, such as adders or counters, and others which are already in simplest prototype form, are identified. In other words, such functions are imbedded within the (trivial) identity transformation.

Section V,A gives some background on alternate approaches to large-scale-integrated (LSI) circuit design. (Some of these approaches are described in more detail in other chapters of this text.) A knowledge of Section IV,A is assumed. The synthesis problem for encoded-input logic is defined in Section V,B. Its solution depends directly on the invariant properties of Fourier transforms from Section II,C. Examples are presented in Section V,C. These examples compare encoded-input logic to alternate approaches within the context of LSI circuit technology.

At its present stage of development, synthesis of encoded-input logic is partly analytic, partly heuristic. The last section (V,D) identifies key mathematical problems whose solutions would remove some of the heuristic aspects of the solution. A knowledge of Section IV is assumed.

A. REVIEW OF LSI LOGIC DESIGN APPROACHES

As background for the encoded input approach to combinational logic design, this section will briefly discuss the impact of LSI technology on logic design criteria and approaches. The LSI circuit technology produced a revolution in methods of manufacturing logic circuitry. Using LSI, large amounts of digital logic (e.g., many hundreds of gates) can be produced as a single LSI array, on a semiconductor substrate of microscopic dimensions (e.g., 0.1 to 0.01 in.²).

There are significant differences between the two major semiconductor technologies, bipolar transistors and metal–oxide–silicon (MOS) in current use. In the interest of brevity, these differences will be ignored in the following comparison, which is relative and not absolute. Generally speaking, the advantages and disadvantages cited below apply to both technologies, although not to the same degree.

1. Classification of LSI Design Approaches

Table XIII classifies LSI design approaches by the nature of their application. The economics of LSI manufacturing is quite sensitive to overall complexity of an array and to production volume. Reliability decreases more than inversely as complexity increases, while design and development costs
TABLE XIII
Classification of LSI Approaches

<table>
<thead>
<tr>
<th>TYPE</th>
<th>UNIVERSAL (MULTIPLE USES PER SUBSYSTEM)</th>
<th>SPECIALIZED (ONE-OF-A-KIND PER SUBSYSTEM)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VARIABLE MEMORY ARRAYS</td>
<td>LOGIC-ORIENTED</td>
</tr>
<tr>
<td></td>
<td>UNIVERSAL LOGIC PACKAGES</td>
<td>MEMORY-ORIENTED</td>
</tr>
<tr>
<td>APPLICATIONS</td>
<td>PROGRAMMABLE DATA BASE REGISTERS</td>
<td>SPECIAL-PURPOSE LOGIC DESIGN WITH TABLE</td>
</tr>
<tr>
<td></td>
<td>FF ARRAYS ASSOCIATIVE MEMORIES</td>
<td>LOOKUP</td>
</tr>
<tr>
<td></td>
<td>DIODE MATRICES AND GATE ARRAYS FOR</td>
<td>HYBRID A/D APPLICATION AND INTERFACE</td>
</tr>
<tr>
<td></td>
<td>SWITCHING, ENCODERS/DECODERS, AND EXTERNALLY-WIRED LOGIC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CIRCUITRY (EXTERNAL OR BORDERS LSI ARRAY)</td>
</tr>
<tr>
<td>DEVICE LAYOUT:</td>
<td>STANDARD</td>
<td>STANDARD</td>
</tr>
<tr>
<td>INTERCONNECTIONS</td>
<td>STANDARD</td>
<td>CUSTOM (IF USED)</td>
</tr>
<tr>
<td>1ST LEVEL (HIGH RESOLUTION)</td>
<td>STANDARD</td>
<td>VARY WITH APPLICATION</td>
</tr>
<tr>
<td>2ND AND 3RD LEVEL (LOW RESOLUTION)</td>
<td>STANDARD</td>
<td>CUSTOM (IF USED)</td>
</tr>
</tbody>
</table>

are three orders of magnitude greater than per unit production costs. Consequently, the first LSI packages to be introduced were “universal” types, which could be used repetitively within a system and sold to more than one customer.

Universal arrays are of two basic types: variable (erasable) memories and universal logic arrays. Memory arrays can be produced in high volume without changing the production tooling, either at the substrate level which defines the layout of solid-state devices (transistor amplifiers or diodes) or at the several levels of deposited wiring for device interconnection. Flip-flop arrays used as registers also include logic for transfer and selection gating. This logic is simple and highly repetitive. Its design is straightforward, and its LSI implementation tends to be pin-limited by the word size used on the parallel bus structure for input/output data transfer.

Generally speaking, it is impossible to provide universal arrays of purely combinational logic (not transfer gating mixed with register memory as described above) of a size and complexity comparable to that of variable memories, without introducing an excessive number of external connections. This is self-defeating from the standpoints of reliability and efficiency of semiconductor utilization. Therefore, varying degrees of specialization are introduced into LSI arrays for combinational logic applications.

The specialized approach to LSI design is shown on the right half of Table XIII. It is subdivided according to the degree to which the device layout and/or interconnections must be tailored to the particular application environment and function. Because specialization reduces production quantities, design, development, and testing costs have a significant effect on unit selling price.
The most highly specialized requirements are imposed by interface circuitry which must satisfy electrical (signal) as well as logical compatibility requirements (e.g., high power output drivers which control external devices and low-level signal detectors (sense amplifiers) in core memories). Packages of this type may be required on the borders of LSI logic arrays. The design of these packages varies so greatly with the application that they cannot be discussed in general terms herein.

The other kind of specialized LSI approach is called a **homogeneous array** in Table XIII. Homogeneous arrays may be either logic-oriented or memory-oriented and are further subdivided in the next section.

### 2. Comparison of Homogeneous Arrays

Table XIV is a more detailed comparison of the memory-oriented approach with three different logic-oriented approaches to homogeneous LSI arrays. These four categories are not mutually exclusive or even partially ordered, as shown later.

Information-theoretic arguments show that for a given $n$, solution complexity is quite sensitive to the size of the smallest subset $f^{-1}(1)$ or $f^{-1}(0)$ to be recognized. All practical techniques must exploit this sensitivity for economical solutions. The number $2^n$ can easily exceed the capacity of the table-lookup (read-only memory) approach. When the size of $f^{-1}(1)$ or $f^{-1}(0)$ is so small that it can be recognized (i.e., its argument vector $x$ can be decoded) at much less cost than storing $2^n$ bits of data (i.e., when the entropy of the problem corresponds to much less than $2^n$ bits of information), then another approach should be used. In principle, both macrocellular and microcellular arrays are “universal” (given enough components). However, a practical approach must find some way to avoid the disjunctive canonical form’s exponential growth factor of $2^n$ while at the same time providing a computationally efficient synthesis algorithm.

The read-only memory approach has a growing number of applications in which the required number of words ($2^n$) does not exceed physical limits on array size and/or speed. (Read-only storage access to one of 2048 bits in 1 µsec, or one of 256 bits in 200 nsec, was available by 1969 using MOS and bipolar LSI technology, respectively, at a cost below $100 in quantity.)

When the topology of the system requires repetitive stages of logic with only a few arguments (e.g., adding and counting circuits, inter-register transfer gates, and address decoders), then the most popular approach is the conventional macrocellular array or its microcellular equivalent. Usually logic for these applications is closely coupled with register memory functions and cannot be simplified further by prototype encoding transformations.
### TABLE XIV
Comparison of Homogeneous Arrays

<table>
<thead>
<tr>
<th>Major Area of Application</th>
<th>Nonrepetitive Logic (Nondecomposable - many inputs, few outputs)</th>
<th>Iterative Arrays (Multistage parallel or sequential operation, with few inputs and outputs per stage)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional or Direct Approach</td>
<td>Read-only Memories (Storage of the truth tables of m output or next-state functions of n arguments)</td>
<td>Macro-Cellular Arrays Custom interconnected flip-flops and Nadu gates for inter-register transfers and/or next-stage mappings)</td>
</tr>
<tr>
<td>State-of-the-Art Limitation</td>
<td>(Device count limited) (m+n), 2^n</td>
<td>I/O or pin-limited Min (PI) {mnw}</td>
</tr>
<tr>
<td>Cost/Size Ratio</td>
<td>Not Needed</td>
<td>Combinatorial Topology</td>
</tr>
<tr>
<td>Synthesis Algorithms</td>
<td>Encoded-Input Logic (Standardized interconnections among cells with limited fanin/fanout)</td>
<td>Micro-Cellular Arrays</td>
</tr>
<tr>
<td>Unconventional or Indirect Approach</td>
<td>(Nand or threshold gates embedded within prototype encoding transformations)</td>
<td>(Device count limited)</td>
</tr>
<tr>
<td>State-of-the-Art Limitation</td>
<td>(I/O or pin-limited) Min {d(n+m)^2 + Min (mnw)} (RAG) {(PI, TF)}</td>
<td>Min {mnw} (CA)</td>
</tr>
<tr>
<td>Cost/Size Ratio</td>
<td>Harmonic Analysis</td>
<td>Canonical</td>
</tr>
<tr>
<td>Synthesis Algorithms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:  
- m = number of function outputs  
- n = number of arguments  
- w = Min \{f^{-1}(1), f^{-1}(0)\}  
- d = average density of encoding matrix  
- PI = Cover by Prime Implicants  
- CA = Cellular Array Groundrules  
- RAG = Restricted Affine Group  
- TF = Cover by Threshold Functions

By suitable definition of cell complexity and array layout, the macrocell and microcellular array approaches can be made to merge into one another. They actually represent points on a spectrum with coarse (macro) structure at one end and fine (micro) structure at the other. There is an inverse relationship between cell complexity and required number of cells, and a similar inverse relationship between interconnection pattern complexity and number of interconnections. The microcellular approach reduces cell size and interconnection pattern to simplest form; it pays for this advantage by a larger number of cells and interconnection paths. Unfortunately, the quantitative dependency of the required number of cells and interconnections on the number and variety of types and intercell connection paths is unknown. Only gross and overly conservative bounds (such as \(n2^n\)) are known for \(\text{min}_{\text{CA}} (mnw)\) (the minimum cost of cellular arrays subject to a particular set (CA) of ground rules). Synthesis techniques which might improve on this bound are not yet available. For an extensive discussion, see Elpas et al., 1967.
3. Encoded Input Logic

This approach to logic design is illustrated in Fig. 7 and 8. The desired function \( f \) is replaced by a simpler function \( g \) imbedded in a prototype encoding transformation. This approach combines both macro- and microcellular approaches in a mathematically tractable way. The prototype encoding transformation uses a regularly connected array of simple cells to reduce a given function with \( n \) inputs and \( m \) outputs to its simplest equivalent form. This imbedded prototype function could be trivial (for example, if the desired function generates the parity check matrix for an error-detecting code). Then, encoded input logic reduces to a microcellular array of 2-input, 1-output cells of a single type (EXCLUSIVE OR gates or mod 2 adders). On the other hand, the encoding transformation itself may be trivial (e.g., the identity matrix); for example, if the desired function already is the simplest member

\[
\begin{align*}
T(x) &= xA \oplus c \\
A &= \begin{cases} a_{i,j} \\
i,j = 1,2,3 \end{cases} \\
y &= xA \oplus c \\
y_i &= \sum x_i a_{i,j} \oplus c_i \\
g(y) &= 0 \text{ or } 1
\end{align*}
\]

FIG. 7. Successive steps in the process of encoding a single output function. (a) Block diagram. (b) Matrix form.
FIG. 8. Equivalence of abstract and schematic representations.

of its prototype equivalence class. In this case, \( g = f \), and the logic reduces to a conventional macrocellular array. Of course, the prototype can be realized as a microcellular array to complete one full circle of inclusion relationships (which shows that these approaches cannot be partially ordered). In particular, logic which is highly repetitive and decomposable (for example, interregister
transfer gating or iterative counting and adding circuits) tends to have macrocellular logic realizations that cannot be improved upon by prototype encoding transformations. The particular example shown in Fig. 8 is for illustrative purposes only. It would not be practical since the cost of encoding outweighs the difference between the normal form for the majority function and its prototype (a 2-input AND gate).

The microcellular approach to LSI can be reduced to linear-input logic but only if additional structure is imposed on one section of the array (the encoding transformation), while at the same time intracell complexity and intercell wiring constraints are relaxed in another direction (to permit macrocellular realization of the embedded function). These qualifications have a very important mathematical consequence. Well-defined synthesis techniques are available both for the encoding transformation (as described herein) and for the embedded prototype function (either of the approaches in the top row of Table XIV can be used). A significant physical implication goes hand in hand with this mathematical tractability: The array size can be strictly bounded by \((n + 1)^2 \min_{RAG} \{d\}\), and the embedded macrocellular array is strictly bounded by \(\min_{RAG} \min_{PL, TF} \{mnw\}\). (To some extent, these two cost elements vary inversely and can be traded against one another.) The notation used here and in Table XIV has this significance: \(\min_{RAG} \{d\}\) indicates that the array density (fraction of EXCLUSIVE OR cells actually utilized as contrasted to mere wiring crossovers) depends on the encoding transformation from within RAG; \(\min_{PL, TF} \{\min_{RAG} \{mnw\}\}\) signifies that prototype encoding transformations can be used to reduce the complexity of the imbedded function. This complexity is characterized exactly by its cubical complex, for which the parameters \(n, w,\) and \(m\) are merely a convenient shorthand. As a practical matter, however, it is fortunate that \(w\) and \(m\) are (empirically) related to the cost of the minimal covering by prime implicants. (In principle, we would like to find the minimum PI or TF cost for every function in the prototype equivalence class, and then select the minimum. However, the number of equivalent functions grows as \(2^n\), and exhaustive search is impossible. Therefore, we are forced to use simpler criteria, such as maximize the correlation between \(f\) or \(f'\) and the \((n + 1)\) Fourier basis functions of zero or one argument variable. Selection of the proper encoding transformation can also be based on reducing the number of levels of threshold logic needed to realize the function. Again, heuristic criteria are used for selection, based on the Fourier coefficients.

Ninomiya (1958) used an empirical criterion for prototype selection in which the Fourier coefficient magnitudes were arranged lexicographically in descending order of their size, after first arranging their argument vectors \(w \in Z^n\) in increasing order of their weight. Unfortunately, the prototype thus selected does not always have a least-cost minimal cover, compared to other possible rearrangements of the transform coefficients, as a counterexample
herein will demonstrate. Perhaps more complicated measures of complexity will produce more accurate relative predictions of the cost of prototype realization. Some possibilities for further research are discussed in Section V,D.

B. SYNTHESIS TECHNIQUES FOR ENCODED INPUT LOGIC

The problem to be considered herein is the following: given a function \( f(x) \) from \( X = Z^n \) to \( Z \), how can we find a simplest realization of the form

\[
f(x) = g(xA \oplus c) \oplus xa' \oplus d
\]  

(113)

This expression is an identity for all \( x \) if and only if \( f \) and \( g \) are in the same prototype equivalence class. In other words, the synthesis problem may be restated as follows: Given \( f \), identity the prototype class to which \( f \) belongs, then find the simplest combination of a prototype class representative \( g \) and encoding transformation components \( A, a, c, \) and \( d \) which satisfy Eq. 113.

A solution to this problem is feasible using the invariant properties of the Fourier transform under prototype transformations. These properties are restated here from the fundamental invariance theorem of Section II,C:

\[
f(x) = g(xA \oplus c) \oplus xa' \oplus d
\]

iff

\[
f_N^*(x) = (-1)^{xa' \oplus d} g(xA \oplus c)
\]  

(114)

iff

\[
g_N^*(w) = (-1)^{wa' \oplus d} f_N^*(wA' \oplus a)
\]

This identity relates the migration of the subset \( f^{-1}(1) \) through the domain \( X \) (and its combination with corresponding subsets of the linear function \( (xa' \oplus c) \)) to the corresponding permutation (or sign changes) of the transform coefficients \( f_N^*(w) \). However, this is useful only to the extent that the coefficient pattern \( f_N^* \) can be related to function complexity. Ninomiya (1958) found empirically that the simplest prototype representative normally had an arrangement of Fourier coefficients \( f_N^*(w) \) which (as far as possible) concentrated the largest coefficient magnitudes at points \( w = e_i \) with unit weight \( |w| = \sum w_j = 1 \). The next two subsections will show that the weight-defining property of \( f^*(0) \) and the implicant-determining property of \( (f^*(0) \pm f^*(e_i)) \) provide not only a rationale for this empirical approach but also an explicit iterative approach for reducing a function to simplest form.

The cubical complex defined by \( f^{-1}(1) \) is invariant to symmetry transformations (argument permutations and complementations). However, prototype transformations have a significant effect on the complexity of the
subcube structure of $f^{-1}(1)$. This is obvious from the fact that prototype transformations compute parity functions, all of whose prime implicants are essential 0-cells or minterms, reducing them to trivial functions of a single variable. Before discussing this effect, the non-weight-preserving property of range translations $(f \oplus xa')$ will be discussed.

1. Altering the Function Weight by Addition of Linear Functions

Prototype transformations have the ability to change the weight of $f$ (size of $f^{-1}(1)$) since they include the addition of linear functions as a subgroup. In this way, a function pair $(f,f')$, both with many minterms (size of $f^{-1}(1)$ approaching $2^n-1$) can often be reduced to another pair $(g,\bar{g})$ with $|g|$ less (or more) and $|\bar{g}|$ more (or less) than $2^n-1$. It has been observed (empirically) that functions with small or large values of $|f|$ are often easier to implement than functions whose weight approaches $2^n-1$.

An algorithm to find the linear function which maximizes or minimizes the weight of $\bar{f}$ or $\bar{f}$ is easily defined:

Let $|f|$ denote the weight of $f$ or cardinality of $f^{-1}(1)$. Then $|f| = f^*(0)$ because all entries are (+1) in the first column of the matrix $Q$ such that $fQ = f^*$. From Section IV.A.6, the first coefficients $f^*(0)$ and $f_N^*(0)$ are related as follows:

\[
f_N^*(0) = 2^n - f^*(0) = 2^n - |f| = -(2^n - |\bar{f}|)
\]

\[
-2^n \leq f_N^*(0) \leq 2^n
\]

In other words, $|f|$ will be reduced to a minimum by maximizing $f_N^*(0)$, and $|f|$ will be maximized by minimizing $f_N^*(0)$.

The next question is how to select a linear function $xa' \oplus d$ which, when added to $f$, produces a function $g$ of minimum or maximum weight. Section IV.A.6 provides the answer. Substituting $w = 0$, we have

\[
g_N^*(0) = 2^n - |g| = -(1)^d f_N^*(a)
\]

That is, $|g| = 2^n - f_N^*(a)$, depending on the value of $d$. In other words, either the maximum or the minimum magnitude of $|g|$ or $|\bar{g}|$ is produced by the same value of $a$, which may be any $w$ such that $f_N^*(w)$ has the largest absolute value.

\[
|\bar{g}|_{\text{max}} = |g|_{\text{max}} = 2^n + \max_a |f_N^*(a)|
\]

Furthermore, which of $g$ or $\bar{g}$ has maximum weight is determined solely by the value of the (arbitrary) binary constant $d$. (This is obvious because $g(x) \oplus d = \bar{g}$ iff $d = 1$.)
2. A 4-Argument Example

The following example shows that adding a linear function can provide a valid alternate to a normal form realization, at least if speed is not a limiting factor.

Figure 9a is the truth table of a function \( f(x) \) with \( n = 4 \), shown as a

<table>
<thead>
<tr>
<th>( f )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>- 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( g )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4 1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8 0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>12 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ F_{ij} \] (The row and column indices \( i \) and \( j \) have \((x_1x_200)\) and \((00x_3x_4)\) for their binary codes, and the \((i,j)\)th entry is \( f(x(i+j)) \)). Figure 9b is the Fourier transform of \( f(x) \), computed as \( Q_2 FQ_2 \). (\( Q_2 \) is defined in Section II,A,4.) A minimal normal form for \( f \) contains 4 prime implicants (three have 3 and one has 4 literals). The total cost of \( f \) is five gates with 17 gate inputs.

From the preceding discussion, it is apparent that we can increase the weight \(|f|\) from 7 = \((2^3 - 1)\) to \((2^3 + 3)\) = 11 by adding the linear function \( xa^t \) to \( f \), where \( a \) is any of the four vectors \( w \) such that \( |f^*(w)| = +3 \). Increasing \(|f|\) may increase the chance of finding larger implicants with four gate inputs. For example, suppose \( a = (1101) \), and define \( g(x) \) as the sum of \( f \) and the linear function \( xa^t \): \( g(x) = f(x) \oplus x_1 \oplus x_2 \oplus x_4 \). The functions \( g \) and \( g^* \) are defined in Figs. 9c and 9d. Notice that \(|g| = g^*(0) = 11 \). Note also that the dot products of the first row and first column of \( g^* \) with \((1, -1, 1, -1)\) and \((1, 1, 1, 1)\), respectively, add up to 16, indicating that at least two 4-element cosets are implicants of \( g \) by the test of Section III,A,3.
The minimal cover of \( g \) contains three prime implicants \( \{(0-0), (0-1), (0-1)\} \). In other words, \( g(x) = \overline{x}_1 \overline{x}_4 \vee \overline{x}_2 \overline{x}_4 \vee \overline{x}_2 x_3 \), which requires only four gates and nine gate inputs. Another three (EXCLUSIVE OR) gates of two inputs each are required to add \( xa^t \) to \( g \); thus, the total cost becomes seven gates with 15 inputs, rather than five gates with 17 inputs. If gate inputs, rather than gate count, is the more important criterion, then the second approach is an improvement over the first one. If a single 4-input EXCLUSIVE OR gate is available, the total cost becomes five gates with 13 gate inputs. For LSI logic, the number of gate inputs is an important factor since it affects the complexity of deposited interconnections and the number of wiring crossovers.

The selection of this particular value of \( a \) was fortuitous. Not all choices reduce \( f \) to such a simple form; some must be followed by a linear transformation \( x \rightarrow xA \oplus c \) on the domain of \( f \), and the total number of gate inputs is greater. As of now, the choice of \( a \) must proceed by trial and error exhaustion of the possibilities.

3. Selection of Linear Domain Transformations

In this section, we give a heuristic technique for selecting a linear transformation which tends to maximize the size and minimize the number of prime implicants in a minimal cover of \( g \) or \( \overline{g} \). A related problem is to select a transformation which minimizes the number and complexity of threshold functions required to realize \( g \). This problem will not be considered herein, but it appears to be a fruitful direction for future research.

The implicant detection theorem of Section III.A.3 provides the basis for heuristic selection. That theorem says

\[
V_b \oplus c \subseteq f^{-1}(z) \quad \text{iff} \quad (f^* \setminus V_b)c^* = 2^n z
\]

(119)

If \( b \) is any subspace of dimension \( n - 1 \), then \( b \) is a unit vector \( e_i \), and \( (f^* \setminus V_b) \) is just the 2-tuple \( \{f^*(0), f^*(e_i)\} \). Its inverse transform has \( f^*(0) \pm f^*(e_i) \) for its two components. Now suppose \( b \) is a vector of weight 2 \( (b = e_i \oplus e_j, 1 \leq i \leq j \leq n) \). Then \( (f^* \setminus V_b) \) is \( \{f^*(0), f^*(e_i), f^*(e_j), f^*(e_i \oplus e_j)\} \), and the four components of its inverse transform are the dot products of this vector with \( (1, 1, 1, 1), (1, -1, 1, -1), (1, 1, -1, -1) \), and \( (1, -1, -1, 1) \), respectively. This clarifies the role of \( f^*(0) \) and \( f^*(e_i) \) in determining prime implicants. An implicant of \( f \) (or \( f \)) with exactly \( k \) literals will exist if and only if one of the \( 2^k \) components of \( (f^* \setminus V_b)^* \) is \( 2^n \) (or 0). Since \( f^*(0) \) enters every such sum, adding \( xa^t \) to \( f(x) \) (thus moving the largest coefficient \( f^*(a) \) to the zero position) tends to maximize the probability that \( g \) or \( \overline{g} \) will possess large implicants.
Further progress toward large implicants can be made by mapping the \( n \) next largest spectral coefficients of \( f_N^*(w \oplus a) \) into the positions \( w = e_i, \ 1 \leq i \leq n \). Let the original positions of these large coefficients be at \( w = a_1, a_2, \ldots, a_n \). To produce \( g_N^*(e_i) = f_N^*(e_i A^i \oplus a) \), it is only necessary to solve the equation

\[
e_i A^i \oplus a = a_i, \quad 1 \leq i \leq n
\]  

(120)

Since \( e_i A^i \) is merely the \( i \)th row of \( A^i \), the matrix \( A \) is defined by

\[
\text{ith row of } A^i = a_i \oplus a, \quad 1 \leq i \leq n
\]  

(121)

In other words, the \( i \)th column of \( A \) is merely the vector \( (a_i \oplus a)^i \), where \( xa^i \) is the linear function which maximized \( g^*_a \), and \( a_i \) is the location of the coefficient \( f^*(w) \) which ranks \( i \)th in magnitude after \( f^*(a) \) itself.

4. A Pathological Example

Before going to a realistic problem, another, more or less pathological example, will be considered. This indicates how a function which would be very costly to implement as a minimal normal form becomes trivial with linear domain encodings.

The classical procedure for simplifying a Boolean expression is to find a minimal normal form, which is a union of prime implicants whose logical sum has the same truth value as the desired function. The effectiveness of this simplification process depends on the nature of the original Boolean expression and the minimization criteria. One example for which this approach runs into serious difficulty is the function represented by the \( 16 \times 8 \) binary array in Table XV. Each column of this array corresponds to one of eight input variables to a function \( f(x) \), and each row of the array represents an assignment of 0 or 1 values to all variables \( x_i \) in the vector \( x \) of input arguments such that \( f(x) = 1 \). For all vectors not listed on the table, \( f(x) = 0 \). This function has 16 essential prime implicants, each one a product of all eight argument variables or their negations. The minimal normal form Boolean expression for \( f(x) \) is the logical sum of these 16 products.

As a matter of practical convenience, the total number of occurrences of the argument variables is often used as a measure of the cost of mechanizing such an expression. For 2-level logic, the number of input signals to individual logic gates is 144.

An alternate representation for the domain of \( f(x) \) in this example is a vector space \( X = \mathbb{Z}^8 \). The 16 vectors in \( f^{-1}(1) \) form a closed subset of this vector space under modulo two addition. In other words, \( f^{-1}(1) \) is a 4-dimensional subspace of \( X \), and our problem becomes one of testing an arbitrary input configuration for membership in this subspace.
TABLE XV
8-Variable Example

<table>
<thead>
<tr>
<th></th>
<th>x_1</th>
<th>x_2</th>
<th>x_3</th>
<th>x_4</th>
<th>x_5</th>
<th>x_6</th>
<th>x_7</th>
<th>x_8</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

To simplify \( f(x) \), we note that there must exist a 4-dimensional subspace \( V \) of \( X \) which is the nullspace of \( f^{-1}(1) \). For this example, \( V = f^{-1}(1) \) (i.e., \( f^{-1}(1) \) is its own nullspace), and each row vector of the table is orthogonal (mod 2) to every other one. Therefore, any linearly independent set of four 8-tuples from the table can be selected as the columns of an \( 8 \times 4 \) binary matrix \( A = \{a_{ij}\} \). The matrix equation \( y = xA \) represents a many-to-one transformation from \( X \) into a space \( Z^4 \) of binary vectors \((y_1, y_2, y_3, y_4)\). This transformation has the unique property that \( y = xA = (0000) \) if and only if \( x \in f^{-1}(1) \). Therefore, for \( y = xA \) define \( g(y) = (\bar{y}_1, \bar{y}_2, \bar{y}_3, \bar{y}_4) \). Then, for any \( x \) in \( X \), \( g(y) = 1 \) if and only if \( f(x) = 1 \). In other words, \( f(x) \) can be mechanized in two steps:

\[
f(x) = g(y(x))
\]

where \( y(x) = xA \) and \( g(y) = (\bar{y}_1, \bar{y}_2, \bar{y}_3, \bar{y}_4) \).
The cost of $g(y)$ is only four gate inputs. Suppose that the circuit complexity of two 2-input mod 2 sum operators or EXCLUSIVE OR gates is comparable to that of one 4-input AND/OR gate. Since each column of $A$ contains four unit entries, each linear equation $y_j = \sum x_i a_{ij} \pmod{2}$ adds six gate inputs (two for each nonzero $a_{ij}$ except the first one in each column). There are many different choices for the matrix $A$, but all of them require 32 gate inputs because row 15 of Table XV must be included in $A$. The total of 28, compared to the original cost of 144, gate inputs represents a very significant reduction in complexity.

C. A 6-VARIABLE EXAMPLE

The following example is a function of six arguments which is the kernel of the 8-variable truth table exhibited on page 79 of Elspas et al. (1967). This truth table defines a universal logic module (ULM) (see Chapter VI), which is capable of realizing any 4-argument function by properly permuting and selecting signal inputs. It has the advantage of being complex and also realistically motivated. The truth table, Fourier transform, and spectrum of this 6-argument function, which has 22 minterms in the subset $f^{-1}(1)$, is shown in Table XVI. A list of prime implicants for $f$ and $\tilde{f}$ is shown in Table XVII. Both tables were generated by a preliminary version of the prime implicant extraction algorithm described in Section III.B. This program was written in the BASIC language for an interactive time-sharing system and is relatively inefficient (e.g., it did not implement the test of Section III.B,4 but searched the entire list of prime implicants to reject redundant ones); 5- or 6-argument problems require 5 or 11 sec, respectively, of central processor time.

A minimal cover was manually generated from the list of 12 PI’s of $f$ (those whose ternary symbol $A$ is followed by $F = 1$ rather than $F = 0$ on Table XVII), plus eight minterms or 0-cells of $f$ that are not covered by any of the PI’s. This cover is defined by Table XVIII; all of the 0-cells and all but the last two 1-cells are essential PI’s. The total number of inputs to the ten AND gates which realizes these terms is $48 + 40 = 88$. Adding the OR gate and its 16 inputs gives 11 gates and 104 gate inputs as a measure of complexity for the 2-level normal form of this function.

The first step in synthesis of encoded input logic is to add a linear function which will not only minimize the weight of $f \oplus xa^t \oplus d$ but also move large coefficient magnitudes into the position $f^*(e_i)$ where $e_i$ is a unit vector. The possible values for $f^*(0) = |f|$ or $2^n - |f|$ are $32 \pm v$, $v = 2, 4, 6, 8, 10$. (From Section IV.A,6, if $h(x) = f(x) \oplus xa^t$, then $2^{n-1} - h^*(0) = h_N^*(0) = f_N^*(a) = -f^*(a)$ for $a \neq 0$.) Thus, $|f| = 22$ or $42$ is the best attainable value,
TABLE XVI
Truth Table, Transform, and Spectrum for 6-Input Example

<table>
<thead>
<tr>
<th>TRUTH TABLE</th>
<th>FOURIER TRANSFORM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROW I J INDICES:</td>
<td>ROW I J INDICES:</td>
</tr>
<tr>
<td>0 1 0 0 1 0 0 0 0 0 0 0</td>
<td>0 22 0 -2 -4 -2 4 -2 0 0 0 0</td>
</tr>
<tr>
<td>8 0 1 1 0 0 0 1 1 1</td>
<td>8 0 2 4 2 4 2 4 2 0 10 1</td>
</tr>
<tr>
<td>16 0 0 1 0 0 0 1 1 1</td>
<td>16 4 2 0 2 8 2 -4 2 2 1 2</td>
</tr>
<tr>
<td>24 0 0 0 0 0 0 1 1 0</td>
<td>24 2 0 2 4 2 -4 2 8 2 2 1</td>
</tr>
<tr>
<td>32 1 0 1 1 0 1 1 0 0</td>
<td>32 -4 -2 -4 2 4 2 4 -2 1</td>
</tr>
<tr>
<td>40 0 0 0 1 0 1 1 0 0</td>
<td>40 -2 0 2 8 2 0 -2 -8 2 2</td>
</tr>
<tr>
<td>48 0 0 0 0 0 0 1 1 0</td>
<td>48 -4 -2 -4 2 -4 6 -4 2 2</td>
</tr>
<tr>
<td>56 0 1 0 0 1 0 0 1 1 3</td>
<td>56 -4 2 4 -2 -4 -2 -4 10 3</td>
</tr>
</tbody>
</table>

SPECTRUM HAS 7 MAGNITUDES

<table>
<thead>
<tr>
<th>SPECTRUM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

and the vector $a$ may be zero or either of the two vectors $w$ where $f^*(w) = 10$ in Table XVI. Choosing $a = (001111)$ will not only retain $|f| = 22$, but it will also interchange the coefficient pairs $(f^*_1, f^*_3), (f^*_2, f^*_4), \text{and} (f^*_8, f^*_6)$, which puts 8, 8, and 10 into the unit vector locations $(i + j) = 16, 32, \text{and their sum} (i + j) = 48$.

Table XIX is the truth table, transform, and spectrum of the sum $f \oplus xa^t$. Note that the spectrum is identical, but the transform coefficient locations have been permuted pairwise by adding a linear function to $f$. Table XX shows that the new function pair $(g, \bar{g})$ has fewer prime implicants (45 instead of 53); furthermore, $f$ now has some 2-cells, and $\bar{f}$ has some 3-cells.
TABLE XVII
Prime Implicants for 6-Input Example

<table>
<thead>
<tr>
<th>1-CELLS (PI's WITH 5 LITERALS)</th>
<th>2-CELLS (PI's WITH 4 LITERALS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P# 28 A=10001- F= 1</td>
<td>P# 1 A=0001- F= 0</td>
</tr>
<tr>
<td>P# 29 A=10100- F= 0</td>
<td>P# 2 A=0110- F= 0</td>
</tr>
<tr>
<td>P# 30 A=10110- F= 1</td>
<td>P# 3 A=1100- F= 0</td>
</tr>
<tr>
<td>P# 31 A=10111- F= 0</td>
<td>P# 4 A=010- F= 0</td>
</tr>
<tr>
<td>P# 32 A=1000-0 F= 1</td>
<td>P# 5 A=00-10- F= 0</td>
</tr>
<tr>
<td>P# 33 A=1010-1 F= 0</td>
<td>P# 6 A=01-00- F= 0</td>
</tr>
<tr>
<td>P# 34 A=000-10 F= 0</td>
<td>P# 7 A=11-01- F= 0</td>
</tr>
<tr>
<td>P# 35 A=011-11 F= 0</td>
<td>P# 8 A=00-1-0 F= 0</td>
</tr>
<tr>
<td>P# 36 A=100-10 F= 1</td>
<td>P# 9 A=01-0-1 F= 0</td>
</tr>
<tr>
<td>P# 37 A=101-11 F= 0</td>
<td>P# 10 A=11-0-0 F= 0</td>
</tr>
<tr>
<td>P# 38 A=110-11 F= 0</td>
<td>P# 11 A=01-00- F= 0</td>
</tr>
<tr>
<td>P# 39 A=111-10 F= 0</td>
<td>P# 12 A=0-010- F= 0</td>
</tr>
<tr>
<td>P# 40 A=10-001 F= 0</td>
<td>P# 13 A=0-01-0 F= 0</td>
</tr>
<tr>
<td>P# 41 A=10-010 F= 1</td>
<td>P# 14 A=0-0-1 F= 0</td>
</tr>
<tr>
<td>P# 42 A=10-101 F= 1</td>
<td>P# 15 A=0-1-00 F= 0</td>
</tr>
<tr>
<td>P# 43 A=10-111 F= 0</td>
<td>P# 16 A=0-100 F= 0</td>
</tr>
<tr>
<td>P# 44 A=1-001 F= 1</td>
<td>P# 17 A=1000- F= 0</td>
</tr>
<tr>
<td>P# 45 A=1-010 F= 1</td>
<td>P# 18 A=1101- F= 0</td>
</tr>
<tr>
<td>P# 46 A=1-011 F= 0</td>
<td>P# 19 A=110-1 F= 0</td>
</tr>
<tr>
<td>P# 47 A=1-100 F= 1</td>
<td>P# 20 A=110-0 F= 0</td>
</tr>
<tr>
<td>P# 48 A=1-110 F= 0</td>
<td>P# 21 A=1-00 F= 0</td>
</tr>
<tr>
<td>P# 49 A=00000 F= 1</td>
<td>P# 22 A=1-000 F= 0</td>
</tr>
<tr>
<td>P# 50 A=00011 F= 1</td>
<td>P# 23 A=1-011 F= 0</td>
</tr>
<tr>
<td>P# 51 A=00111 F= 0</td>
<td>P# 24 A=0-001 F= 0</td>
</tr>
<tr>
<td>P# 52 A=01010 F= 1</td>
<td>P# 25 A=0-100 F= 0</td>
</tr>
<tr>
<td>P# 53 A=01110 F= 0</td>
<td>P# 26 A=10000 F= 0</td>
</tr>
<tr>
<td>28 10001-</td>
<td>27 A=1111</td>
</tr>
</tbody>
</table>

TABLE XVIII
Minimal Cover by Prime Implicants

<table>
<thead>
<tr>
<th>PI Number</th>
<th>One-Cells</th>
<th>0-Cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>-00000</td>
<td>001001</td>
</tr>
<tr>
<td>50</td>
<td>-00011</td>
<td>001111</td>
</tr>
<tr>
<td>52</td>
<td>-01010</td>
<td>010010</td>
</tr>
<tr>
<td>44</td>
<td>1-0101</td>
<td>010111</td>
</tr>
<tr>
<td>45</td>
<td>1-0110</td>
<td>011101</td>
</tr>
<tr>
<td>47</td>
<td>1-1100</td>
<td>011110</td>
</tr>
<tr>
<td>42</td>
<td>10-101</td>
<td>111001</td>
</tr>
<tr>
<td>28</td>
<td>10001-</td>
<td>111111</td>
</tr>
</tbody>
</table>
### TABLE XIX
Truth Table, Transform, and Spectrum after Adding Linear Function

<table>
<thead>
<tr>
<th>TRUTH TABLE:</th>
<th>( \mathbf{J} ) INDICES:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i )</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>48</td>
<td>1</td>
</tr>
<tr>
<td>56</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FOURIER TRANSFORM:</th>
<th>( \mathbf{J} ) INDICES:</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i )</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>-8</td>
</tr>
<tr>
<td>24</td>
<td>-2</td>
</tr>
<tr>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>40</td>
<td>-4</td>
</tr>
<tr>
<td>48</td>
<td>-10</td>
</tr>
<tr>
<td>56</td>
<td>4</td>
</tr>
</tbody>
</table>

SPECTRUM HAS 7 MAGNITUDES

<table>
<thead>
<tr>
<th>SPECTRUM:</th>
</tr>
</thead>
<tbody>
<tr>
<td>INDEX</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

The positions of the larger coefficients can be further improved by the linear transformation \( y = xA \) with \( A \) defined below.

\[
A = \begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]  

(123)

This matrix \( A \) was chosen by requiring \( A^t \) to permute certain coordinates of \( f^* \). Namely, \( A^t \) leaves invariant the unit vectors \( e_1, e_2, \) and \( e_5 \) (w(i) for \( i = 32, 16, \) and 2) and interchanges 4 with 36. In order to make \( A^t \) nonsingular
<table>
<thead>
<tr>
<th>TABLE XX</th>
<th>Prime Implicant Table after Adding Linear Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-CELLS (PI'S WITH 1 LITERALS)</td>
<td></td>
</tr>
<tr>
<td>4-CELLS (PI'S WITH 2 LITERALS)</td>
<td></td>
</tr>
<tr>
<td>3-CELLS (PI'S WITH 3 LITERALS)</td>
<td></td>
</tr>
<tr>
<td>PI #</td>
<td>Value</td>
</tr>
<tr>
<td>1</td>
<td>000-0</td>
</tr>
<tr>
<td>2</td>
<td>11-1</td>
</tr>
<tr>
<td>3</td>
<td>1-01</td>
</tr>
<tr>
<td>4</td>
<td>1-1-0</td>
</tr>
<tr>
<td>2-CELLS (PI'S WITH 4 LITERALS)</td>
<td></td>
</tr>
<tr>
<td>PI #</td>
<td>Value</td>
</tr>
<tr>
<td>5</td>
<td>0111</td>
</tr>
<tr>
<td>6</td>
<td>001-1</td>
</tr>
<tr>
<td>7</td>
<td>010-1</td>
</tr>
<tr>
<td>8</td>
<td>100-0</td>
</tr>
<tr>
<td>9</td>
<td>11-0</td>
</tr>
<tr>
<td>10</td>
<td>001-1</td>
</tr>
<tr>
<td>11</td>
<td>100-1</td>
</tr>
<tr>
<td>12</td>
<td>101-0</td>
</tr>
<tr>
<td>13</td>
<td>111-1</td>
</tr>
<tr>
<td>14</td>
<td>01-11</td>
</tr>
<tr>
<td>15</td>
<td>01-1-1</td>
</tr>
<tr>
<td>16</td>
<td>00-11</td>
</tr>
<tr>
<td>17</td>
<td>01-10</td>
</tr>
<tr>
<td>18</td>
<td>10-00</td>
</tr>
<tr>
<td>19</td>
<td>11-01</td>
</tr>
<tr>
<td>20</td>
<td>1-0-01</td>
</tr>
<tr>
<td>21</td>
<td>1-1-00</td>
</tr>
<tr>
<td>22</td>
<td>0-0000</td>
</tr>
<tr>
<td>23</td>
<td>0-0101</td>
</tr>
<tr>
<td>24</td>
<td>0-0001</td>
</tr>
<tr>
<td>25</td>
<td>0-0100</td>
</tr>
<tr>
<td>26</td>
<td>0-0000</td>
</tr>
<tr>
<td>27</td>
<td>0-0011</td>
</tr>
<tr>
<td>28</td>
<td>0-0110</td>
</tr>
<tr>
<td>29</td>
<td>0-0000</td>
</tr>
<tr>
<td>30</td>
<td>0-0011</td>
</tr>
<tr>
<td>31</td>
<td>0-0001</td>
</tr>
<tr>
<td>32</td>
<td>0-0100</td>
</tr>
<tr>
<td>33</td>
<td>0-10000</td>
</tr>
<tr>
<td>34</td>
<td>0-1011</td>
</tr>
<tr>
<td>1-CELLS (PI'S WITH 5 LITERALS)</td>
<td></td>
</tr>
<tr>
<td>PI #</td>
<td>Value</td>
</tr>
<tr>
<td>35</td>
<td>0100-0</td>
</tr>
<tr>
<td>36</td>
<td>1011-1</td>
</tr>
<tr>
<td>37</td>
<td>011-01</td>
</tr>
<tr>
<td>38</td>
<td>101-01</td>
</tr>
<tr>
<td>39</td>
<td>110-10</td>
</tr>
<tr>
<td>40</td>
<td>0-0101</td>
</tr>
<tr>
<td>41</td>
<td>0-0110</td>
</tr>
<tr>
<td>42</td>
<td>0-1100</td>
</tr>
<tr>
<td>43</td>
<td>0-10000</td>
</tr>
<tr>
<td>44</td>
<td>0-1011</td>
</tr>
<tr>
<td>45</td>
<td>0-1101</td>
</tr>
</tbody>
</table>

214
and triangular, rows 3 and 6 were also chosen to interchange 8 with 24 and 1 with 41 (coefficients of \( f^* \)). The most obvious effect of \( \bar{A}^t \) is to replace \( f^*(e_3) = 2 \) by \( g^*(e_3) = h^*(e_3 \bar{A}^t) = h^*(w(36)) = 8 \). In this way we obtain still a third truth table for \( g(y) \) with \( y = x \bar{A}^t, g(y) = h(x) = f(x) \oplus xa^t \) for all \( x \), and a transform \( g^*(w) = h^*(w \bar{A}^t) (g^*(e_i) = h^* \) evaluated at \((i\text{th row of } \bar{A}^t)\) (see Table XXI). Table XXII shows that a further simplification has been achieved (\( g \) has eleven 3-cells, \( h \) had only four). Only 5 PI's are essential now, and only 1 of them is a 0-cell. After eliminating points covered by the essential PI's, other PI's become essential. The minimal cover in Table XXIII requires only 8 AND gates with 37 inputs.

**TABLE XXI**

Truth Table, Transform, and Spectrum for Encoded Input Logic

### TRUTH TABLE OF \( F(X) \):

<table>
<thead>
<tr>
<th>ROW</th>
<th>I</th>
<th>J</th>
<th>Indices:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

### TRANSFORM OF \( F(X) \):

<table>
<thead>
<tr>
<th>ROW</th>
<th>I</th>
<th>J</th>
<th>Indices:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>22</td>
<td>-4</td>
<td>-2</td>
<td>-2</td>
<td>-4</td>
<td>-8</td>
<td>2</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>-2</td>
<td>4</td>
<td>-2</td>
<td>4</td>
<td>-4</td>
<td>-2</td>
<td>4</td>
<td>-2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>-2</td>
<td>4</td>
<td>-2</td>
<td>4</td>
<td>-4</td>
<td>-2</td>
<td>4</td>
<td>-2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>2</td>
<td>-4</td>
<td>-2</td>
<td>-2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>-2</td>
<td>-2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>-2</td>
<td>4</td>
<td>-4</td>
<td>-2</td>
<td>4</td>
<td>-2</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>-10</td>
<td>4</td>
<td>2</td>
<td>-8</td>
<td>-4</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>2</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>4</td>
<td>-2</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SPECTRUM HAS 7 MAGNITUDES:

<table>
<thead>
<tr>
<th>INDEX</th>
<th>ABSEV</th>
<th>POS</th>
<th>NEG</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>12</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>10</td>
<td>18</td>
<td>28</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

END OF FXFM & SPCTRNM
**TABLE XXII**
Prime Implicant Table for Encoded Input Logic

### 5-CELLS (PI's WITH 1 LITERALS)

<table>
<thead>
<tr>
<th>PI#</th>
<th>A</th>
<th>F =</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>03</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>04</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>06</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### 4-CELLS (PI's WITH 2 LITERALS)

<table>
<thead>
<tr>
<th>PI#</th>
<th>A</th>
<th>F =</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>02</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>03</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>04</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>05</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>06</td>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>07</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>08</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>09</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1 1</td>
<td>0</td>
</tr>
</tbody>
</table>

### 3-CELLS (PI's WITH 3 LITERALS)

<table>
<thead>
<tr>
<th>PI#</th>
<th>A</th>
<th>F =</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>02</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>05</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>07</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>08</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>09</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

### 2-CELLS (PI's WITH 4 LITERALS)

<table>
<thead>
<tr>
<th>PI#</th>
<th>A</th>
<th>F =</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>02</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>05</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>07</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>08</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>09</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1 0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1 0</td>
<td>1</td>
</tr>
</tbody>
</table>

### 1-CELLS (PI's WITH 5 LITERALS)

<table>
<thead>
<tr>
<th>PI#</th>
<th>A</th>
<th>F =</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>02</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>03</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>04</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>05</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>06</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>07</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>08</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>09</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1 1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

---

*Note: The table lists the prime implicants for a 3-variable Boolean function, with each line indicating a PI and its corresponding truth value.*
TABLE XXIII
Minimal Cover of $g(y')$

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 - 1 1</td>
<td>1 0 1 0 - 1</td>
<td></td>
</tr>
<tr>
<td>0 - 1 0 1</td>
<td>0 1 0 - 0 0</td>
<td></td>
</tr>
<tr>
<td>- 1 - 1 0</td>
<td>0 - 0 1 1 0</td>
<td></td>
</tr>
<tr>
<td>- - 1 1 0</td>
<td>1 0 0 1 1 1</td>
<td></td>
</tr>
</tbody>
</table>

Four EXCLUSIVE OR gates (8 inputs) for $xa^t$ and four more (8 inputs) for $A$ gives a total of 16 gates and 53 inputs rather than 11 gates with 104 gate inputs for the original function. Clearly, which realization is preferred depends on the relative weighting assigned to number of gates versus gate inputs. However, gate inputs tend to be more important in LSI because they correspond to signal paths (deposited wiring), and a decrease in their number is likely to correspond to a decrease in the number of wiring crossovers as well.

The conjecture that minimum-cost prototypes are produced by mapping larger coefficient magnitudes $g^*(w)$ into locations with smallest weight $|w| = \sum w_i$ is disproved by the following counterexample. The distribution of spectrum coefficients of the three largest magnitudes $|g^*(w)|$ on Table XXI is shown below versus the weight of the argument $w$.

| $|g^*(w)|$ | $\{w\}$ | $|w|$ |
|---------|--------|------|
| 22      | 0      | 0    |
| 10      | 7, 48  | 2, 3 |
| 8       | 4, 16, 32, 51 | 1, 1, 1, 4 |
| 6       | 17     | 2    |

We now construct a further mapping of $g$ into a new prototype. It is obvious that the two largest coefficients, $|g^*| = 10$, can be relocated to locations of weight 1 (say, to $w = 16, 32$) by selecting the first two rows of a new matrix $A^t$. After this is done, it is possible to select only two more linearly independent $A$-rows from among the 4 vectors $w$ where $|g^*(w)| = 8$. These vectors (indices 32, 4) become the third and fourth rows of $A^t$. Finally, the vector $w = 17$ for which $|g^*(w)| = 6$ becomes row 5, and $w = 40$ for which $|g^*(w)| = 4$, becomes its last row. The resulting matrix $A_2$ will encode $y$ as $z = yA_2$ and supply inputs to a new prototype function $p(z)$ such that $p(yA_2) = g(y) = f(x)$ for $z = yA_2 = xAA_2, x \in \mathbb{Z}^n$. The minterms of $p$ are as follows:
p^{-1}(1) = \{4, 6, 7, 12, 14-23, 26, 30, 51-54, 58, 63\}. The matrix \( A_2 \) and the locations of large transform coefficients \( p^*(w) \) are as follows:

\[
A_2 = \begin{bmatrix}
0 & 1 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\]

\[
\begin{array}{ccc}
p^*(w) & \{w\} & |w| \\
10 & 16, 32 & 1, 1 \\
8 & 4, 8, 24, 52 & 1, 1, 2, 4 \\
6 & 2 & 1 \\
4 & 1, 10, \ldots & 1, 2, \ldots
\end{array}
\]

A comparison of \( g^*(w) \) to \( p_*(w) \) for \(|w| = 1\) is given below:

\[
\begin{array}{ccccccc}
w & 32 & 16 & 8 & 4 & 2 & 1 \\
|g^*(w)| & 8 & 8 & 2 & 8 & 2 & 4 \\
|p^*(w)| & 10 & 10 & 8 & 8 & 6 & 4
\end{array}
\]

Clearly, \( p^* \) achieves a better score by this empirical measure of complexity (largest coefficient values at vectors \( w \) of lowest weight). Unfortunately, however, the minimal covering of \( p(z) \) turns out to be more costly than that for \( g(y) \) in Table XXII above, in spite of the fact that \( p \) and \( \overline{p} \) together have only 24 PI's, two of which are dimension 4, while \( g \) and \( \overline{g} \) have 36 PI's, all of dimension 3 or less (see Table XXI). Specifically, a minimal two-level cover for \( p(z) \) or \( \overline{p}(z) \) requires 48 or 43 gate inputs, respectively, while \( g(y) \) can be covered by a two-level form with only 37 gate inputs.

**D. FUNDAMENTAL PROBLEMS AND EXTENSIONS**

One cannot fail to be impressed with the conceptual insight and unifying principles provided by harmonic analysis. It provides a common basis for hitherto unrelated techniques for implicant extraction, functional decomposition, and the analysis of linear encodings. Unfortunately, the theory developed in this Chapter can be directly applied only to the space \( \mathcal{F} \) of all single-output logic functions. Consequently, extensions to multiple-output functions, and then to sequential machines or finite state automata have great
potential value. This section identifies several key problem areas which are likely candidates for further research, namely, encoded input threshold logic synthesis, the probability distribution of spectral coefficient magnitudes for randomly chosen functions, canonical forms for elements of RAG(n/m) with m > 1, multiple-output function synthesis, and applications of harmonic analysis to sequential circuits.

1. Synthesis Techniques for Encoded Input Threshold Logic

Recent literature on threshold logic makes an increasing number of references to the Hadamard (abstract Fourier) transform basis for \( \mathbb{Z}^n \) (Hwa and Sheng, 1969). However, no one has yet attempted to use prototype transformations to reduce a function (as much as possible) toward a threshold-realizable form. Consequently, the utility of Fourier transforms as a tool to aid in this reduction has also gone unrecognized. Perhaps the invariant properties of the Fourier transform and its implicant detection capability will lead to an algorithm for extraction of linearly separable subfunctions and for an encoding transformation which identifies the simplest threshold-realizable prototype class representatives.

2. Probability Distribution of Spectral Coefficient Values

Section IV.A.6 showed that adding \( xa^i \) to \( f(\mod 2) \) produces a function \( g \) whose weight is \( f^*(a) \) rather than \( f^*(0) \). A smaller or larger weight (number of points in \( f^{-1}(1) \)) often leads to a simpler realization of \( f \); therefore, the largest magnitude \( f^*(w) \), or more generally, the distribution of spectral coefficient magnitudes, is of interest for a randomly chosen function. It may be that large weight reduction factors cannot be expected for randomly chosen functions. An intuitive argument might be based on an analysis which shows that there are neither enough functions of low weight to go around, nor enough linear functions to add, as \( n \) becomes large.

If \( f(x) = 0 \) or 1 with probability \( \frac{1}{2} \), then an integer-valued spectral coefficient \( f_n^*(w) \) is the sum of \( 2^n \) variables with mean 0 and variance \( (\frac{1}{4}) \). Therefore, the standard deviation of \( f_n^*(w) \) is \( 2^{(n/2)-1} \). In other words, \( f_n^*(w) \) coefficients tend to cluster within a fraction \( 2^{-n/2} \) of the midpoint of their range \( \pm 2^{n-1} \). Another approach to this problem would be to use order statistics (probability that the largest of \( 2^n \) Fourier coefficients has a certain value). More generally, the distribution of the \( n + 1 \) largest coefficient values is of interest to evaluate the general effectiveness of the techniques of Section V.B.
3. Canonical Forms for Elements of RAG(n/m)

The constructive definition of primary rational canonical forms for prototype transformations appears to be a fundamental unsolved problem of affine (and linear) group theory when the number of output variables is generalized from 1 to \( m \). The problem can be stated mathematically as follows: Given a subgroup RAG(n/m) of the general affine group, find a canonical representative for each equivalence class of matrices in RAG(n/m) under similarity transformations which are inner automorphisms of RAG(n/m). The single constraint on this subgroup is that it leaves invariant a given \( m \)-dimensional subspace of the space \( \mathbb{Z}^{n+m} \) on which it acts. This constraint generalizes the \( b = 0 \) restriction in the derivation of RAG(n/1) in Section IV,A,2. For \( m = 1 \), unique canonical forms were derived for prototype transformations by Lechner (1963). However, this was a specialized construction which assumed an output space of one dimension.

4. Multiple Output and Many-Valued Functions

In Section III, prime implicants, essential prime implicants, and decomposability conditions were identified for a single-output function using harmonic analysis. One powerful generalization of harmonic analysis techniques would be their extension to \( m \)-output combinational logic. Of course, this means generalizing the group RAG to RAG(n/m) as described in Section IV,A,4.

Of course, the covering problem for multiple-output functions can apply the technique of Section III one function at a time. More generally, since RAG(n/m) includes AG(m) on the space of output \( m \)-tuples \( f = (f_1, \ldots, f_m) \), \( 2^m \) different functions can be produced from any linearly independent subset of \( m \) functions \( (g_1, \ldots, g_m) \). For \( m \) sufficiently small, it is possible to evaluate the cost of each of the \( 2^m \) linear combinations of \( f_1 \) through \( f_m \), then select a basis \( g = g_1, \ldots, g_m \) for this subspace of \( \mathcal{F} \), and a mapping \( B \) such that \( gB = f \) is an identity for every argument \( x \). Of course, this technique is also applicable to a \( 2^m \)-valued function under a suitable coding of its outputs into binary \( m \)-tuples.

One question of interest here, particularly if \( n \) is larger than \( m \), is whether some of the functions to be implemented are actually modulo two linear combinations of other functions that can be realized in a more efficient manner. Thus, we might realize a set of functions \( g_1, \ldots, g_m \), then apply an \( m \times m \) affine transformation to these \( m \) functions and produce \( f_1, \ldots, f_m \). In matrix form, \( f = gB \oplus d \), with \( B \in LG(m) \) and \( d \in \mathbb{Z}^m \). The prototype transformation group RAG(n/m) includes such transformations on the outputs of
imbedded prototype functions \( g_1 \) through \( g_m \). A straightforward generalization will handle functions \( f_1, \ldots, f_m \) of rank \( k < m \) by \( k \) imbedded functions \( g_1, \ldots, g_k \).

An exhaustive evaluation of the \( 2^m \) candidate basis functions is possible for reasonable values of \( m \) (say, \( m \leq 10 \)). The evaluation would proceed by analyzing the Fourier spectrum (or even solving the minimal covering problem if \( m \) is small enough) for each of the \( 2^m \) possible linear combinations (mod 2) of the functions \( f_1, \ldots, f_m \) (regarded as \( 2^m \)-tuples over \( \mathbb{Z} \)). From among these \( 2^m \) functions, a basis \( (g_1, \ldots, g_m) \) of lowest cost would be selected. The only restriction on this basis is that (as \( 2^m \)-tuples over \( \mathbb{Z} \)) they have the same rank \( k \leq m \) as the original set of functions \( f_1, \ldots, f_m \). In other words, this multiple-output synthesis approach works backward from the required \( m \) functions and attempts to find \( k \leq m \) other functions, which may belong to other prototype classes. Then a least-cost representative of each prototype class is mechanized, and finally the domain encodings which supply input variables to these class representatives are combined and redundant columns of their respective matrix representation are discarded. This technique appears worthwhile even though it does not represent a general solution to the problem in the group theoretic sense.

5. The State Assignment Problem for Sequential Circuits

A synchronous sequential circuit or finite-state automata is a 6-tuple \( M = (S, I, \emptyset, T, f, g) \) in which \( I, S, \) and \( \emptyset \) represent the space of inputs \( X = \mathbb{Z}^k \), states \( S = \mathbb{Z}^m \), and outputs \( U = \mathbb{Z}^p \), respectively, (Hartmanis and Stearns, 1966). For convenience, we also identify \( T \) as the space of “next state” values and assume that \( T \) is mapped back into \( S \) by the identity transformation after a finite time delay. The functions \( f(X \oplus S \rightarrow T) \) and \( g(X \oplus S \rightarrow U) \) are identical to \( \delta \) and \( \lambda \) of Hartmanis and Stearns (1966) and define the next-state and output mappings, respectively. As usual, when the sequential structure of \( f \) is being analyzed, we concentrate on \( f \) and ignore \( g \). The next-state mapping \( f \) for a sequential circuit with \( k \) inputs, \( p \) outputs, and \( m \) state variables is defined by an \( m \)-output combinational logic function of \( n = m + k \) arguments. Therefore, harmonic analysis may be a useful synthesis tool provided that the theory presented in this chapter can be generalized to multi-output or vector-valued functions.

In their text, Hartmanis and Stearns (1966) pointed out that isomorphic machines are identical except for renaming the states, inputs, and outputs. The group \( RAG(k/m/m) \) defined in Section IV,A.4 is a subgroup of the permutation subgroup on \( (X \oplus S \oplus T) \) which induces the “isomorphic machine” equivalence relation. Therefore, each class of isomorphic machines is a union of prototype equivalence classes.
Suppose that we start with a completely arbitrary state assignment and attempt to decompose the permutation of \((X \oplus S \oplus T)\) which produces the "best" equivalent machine (best reassignment of state names) into two parts: (1) a recoding of the initial assignment into a coset of the prototype class which contains the "best" assignment and (2) a member of RAG which produces the "best" assignment. The second map has a well-defined algebraic structure, and harmonic analysis can be used as a synthesis tool. The first map still requires the state partitioning techniques of Hartmanis and Stearns (1966).

One important question is whether the decomposability properties of the next state mapping are more sensitive (less invariant) to the between-coset mappings or to the intracoset prototype transformation. If decomposability is more affected by mappings between cosets, then harmonic analysis complements state partitioning techniques and should follow them in the synthesis procedure. On the other hand, if decomposability is more sensitive to elements of RAG, then arbitrary state coding followed directly by harmonic analysis may replace current state assignment techniques as a direct synthesis approach. The example of Section IV.C produced a triangular matrix \(A\) which corresponds to a cascade-type realization. This suggests that RAG does affect decomposability of next state mappings.

6. Orbits and Stability Groups

Generally speaking, a function \(f\) which possesses symmetries under some reasonable transformation group on \(X\) is simpler to realize than a function which does not. The same is true for functions which are invariant to certain elements of \(\text{AG}(n)\) on \(X\). These elements must permute the subsets \(f^{-1}(1)\) and \(f^{-1}(0)\) within themselves (without mixing them up). Actually, RAG permutes the corresponding subsets of the characteristic function of \(f\) as a subset of \(X \oplus Z\)—see Section IV.A.2. However, for simplicity, we will consider only \(\text{AG}(n)\) rather than \(\text{RAG}(n/1)\) herein.

The set of all elements of \(\text{AG}(n)\) which leave a given function \(f\) invariant form a group, called the stability group of \(f\) under \(\text{AG}(n)\), denoted \(G_f\). For notational simplicity, the terminology and properties of the partition lattice theory as described by Hartmanis (1960) and two other concepts "level set" and "orbit" borrowed from topological dynamics (via Bellerman, 1961) will be used in this paragraph.

Each element \(t\) in \(G_f\) generates a cyclic subgroup which permutes the set \(X \oplus Z\); the cycle sets of this permutation define a partition of \(X\) denoted \(\pi_t\). The operation of combining overlapping cycles from all partitions \(\pi_t, t \in G_f\),
produces the lattice-theoretic join, or upper bound of the partitions \( \pi_i \); this join is denoted \( \pi_f^G \), and the disjoint subsets of \( X \) which are its components are called orbits of \( G_f \). Note that \( \pi_t \leq \pi_f^G \) for all \( t \in G_f \) (\( \leq \) denotes lattice inclusion here). Two elements \( x \) and \( y \) of \( X \) are in the same orbit of \( G_f \) iff some element of \( G_f \) maps \( x \) into \( y \).

The two subsets \( f^{-1}(1) \) and \( f^{-1}(0) \) are called level sets of \( f \). They also define a partition of \( X \), called the level set partition and denoted \( \pi_f^L \). These two partitions induced by \( f \) are fundamentally related as follows: \( \pi_f^G \leq \pi_f^L \).

It is known that similarity transformations preserve the lattice of invariant subspaces of a subgroup of \( \text{LG}(n) \) (Thrall, 1952). Examples indicate that a similarity transformation on \( \text{AG}(n) \) which maps one or more generators of \( G_f \) into rational canonical form is equivalent to an encoding transformation which maps \( f \) into a prototype class representative \( g \) of "simplest" form (Lechner, 1963a,b). The heuristic explanation for this is that a canonical matrix in \( \text{LG}(1/n) \) (isomorphic to \( \text{AG}(n) \)) has a quasidiagonal form, and therefore the invariant subspaces of this form have bases consisting of unit vectors. Since \( \pi_g^G \leq \pi_g^L \), the prototype \( g \) corresponding to \( f \) must be constant on the orbits of \( \pi_g^G \) whose unions are these subspaces. We conjecture that more prime implicants are likely to exist for \( g \) than for any other member of the prototype class and that this holds true in general.

In a short paper on coding theory, Wells (1960) showed how harmonic analysis could be used to successively refine \( \pi_f^L \) so that the sequence of resulting partitions of \( X \) had \( \pi_f^G \) for their lower bound. (The invariance theorems of Section II,C show that \( \pi_f^L \), the level set partition of \( f^* \), and \( \pi_f^G \), the stability group partition of \( G_f \), are intimately related.) Lechner (1963a,b) showed that the orbits of \( G_f \) provide useful information which helps to identify the group \( G_g \). Although Well's algorithm never became useful in the coding theory context, we conjecture that it can be extended to affine groups and will then have far reaching theoretical and practical consequences in switching theory as an aid to prototype class identification and synthesis of encoded input logic.

7. Simplification of Encoding Transformations

One important tradeoff in synthesis of encoded-input logic can be posed as follows: How does \( d \) (the density of 1's in the encoding transformation \( t \in \text{RAG} \)) vary with the complexity of the prototype class representative \( g \) chosen to implement the function? One extreme is to imbed a unique prototype \( g = ft \), in which case the encoding transformation \( t \) can still be varied within the coset \( itG_g \) of the stability group of \( g \) defined in Section V,D,6.
The opposite extreme is to permit a prototype representative from any of the symmetry types in the prototype equivalence class. For one of these symmetry types (the one to which \( f \) belongs), the encoding transformation of the input arguments reduces to a (trivial) permutation matrix. The examples of Sections V.B.4 and V.C show that a relatively simple encoding transformation (a sparse matrix with only 15 out of a possible 51 unit elements) can achieve impressive results in some cases.

A similar problem can be posed for multiple-output functions: How many distinct columns are needed in the matrix \( A \) in order to transform \( x \) into the most appropriate vectors \( y_1, \ldots, y_m \) such that \( m \) different functions \( f_1 \) to \( f_m \) can be realized from \( g_1(y_1) \) through \( g_m(y_m) \)? One extreme is to restrict \( A \) to \( n \) columns, in which case only one of the \( m \) functions may be completely reducible to prototype form; the other extreme is to transform each function independently to prototypes \( g_1, \ldots, g_m \), by elements \( t_1 \) through \( t_m \) in RAG, then search through the cosets \( t_1G_{g_1}, \ldots, t_mG_{g_m} \) of the stability groups of \( g_i \) through \( g_m \) to find the combination of \( A \)-matrices with the smallest total number of unit entries or the smallest number of distinct columns.

**EXERCISES**

16. Derive an equivalent algebraic formula for the majority function illustrated on Fig. 8 in terms of the input arguments \( x_1, x_2, \) and \( x_3 \) and the imbedded function \( g(y) = y_1y_2 \). Define \( A, a, c, \) and \( d \) for the encoding transformation relating \( f \) to \( g \). Does the option of inputting complemented or uncomplemented arguments affect the cost of the encoding transformation?

17. Derive another equivalent formula for the majority logic function of Fig. 8 using \( x_1, x_2, \) and \( x_3 \) as inputs with a different prototype function \( g(y) = y_1 \lor y_2 \). Define the components \( A, a, c, \) and \( d \) of the encoding transformation. Does the prototype selection influence the cost of the encoding transformation?

18. Starting with the function \( g(y) \) in Fig. 15, find a matrix \( A \) such that \( z = yA, p(z) = g(y), \) and \( p^*(w) = 3 \) at \( w = 1, 2, 4, 8, \) and 15.

19. Starting with the function \( f(x) \) in Fig. 9, find a matrix \( A^1 \) such that \( h^*(w) = f^*(wA^1) = \pm 3 \) for \( w = w(i), i = 1, 2, 4, 8, 6, 9, f(x) = h(xA), y = xA, \) and \( h(y) = y_1y_2 \lor y_1\bar{y}_3 \lor \bar{y}_2\bar{y}_4 \lor \bar{y}_3\bar{y}_4 \).
REFERENCES


