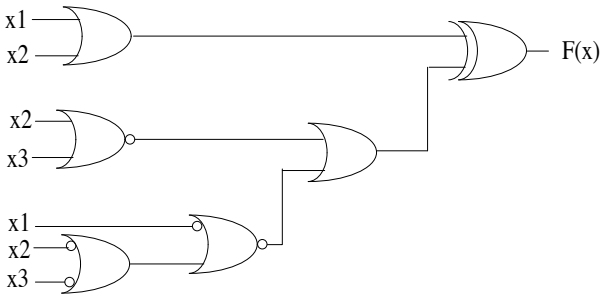

Figure 6: Final Circuit using Multi-Level Design Process

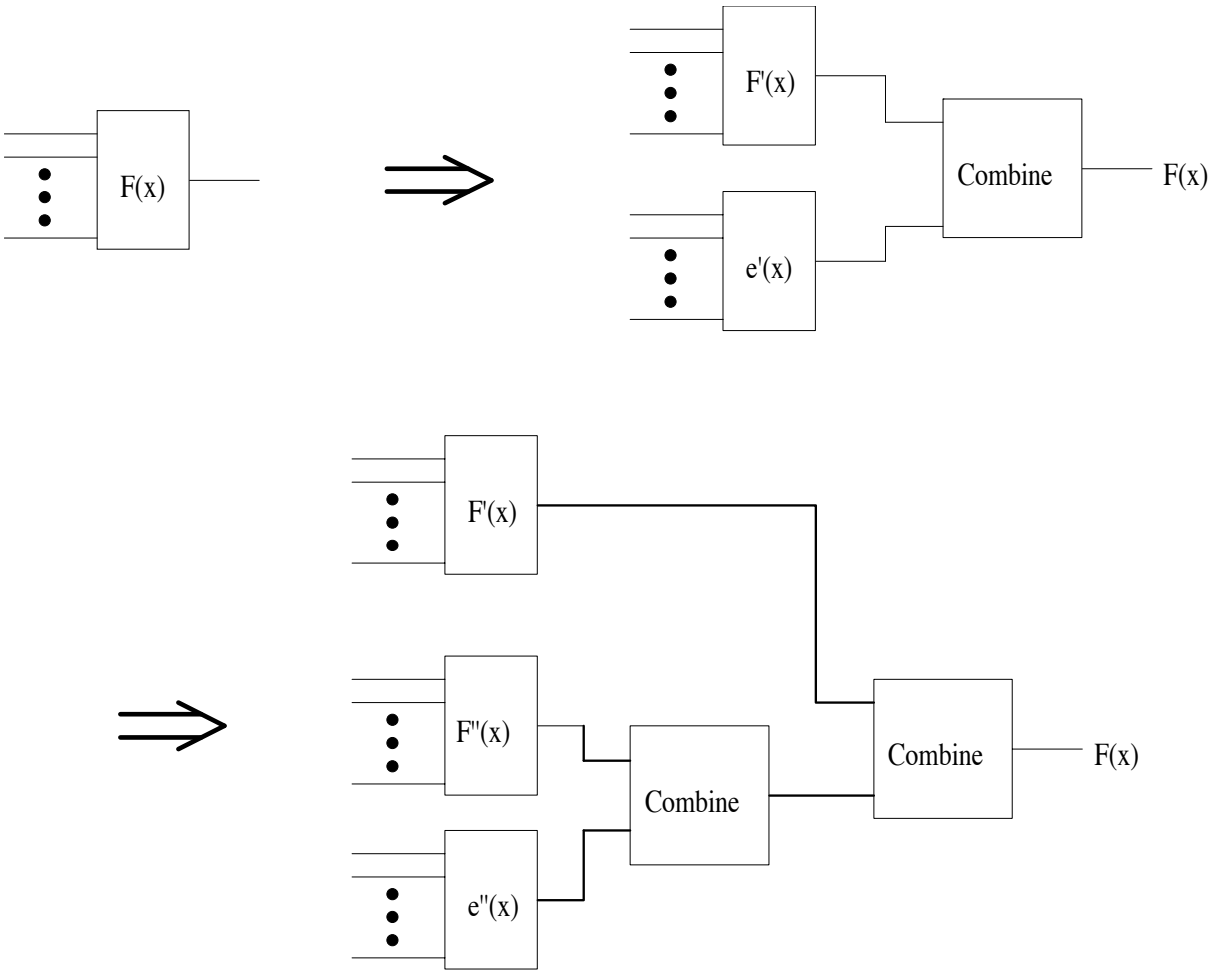

Figure 3: Diagram of Multi-Level Synthesis Technique

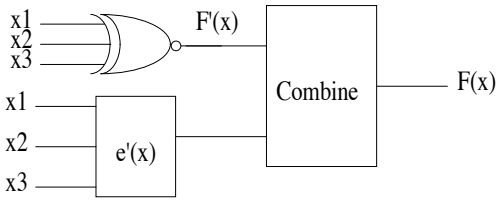

Figure 4: First Iteration of Two-Level Synthesis of Example "Function"

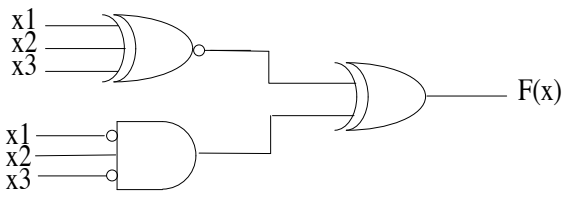

Figure 5: Final Circuit using Two-Level Design Process

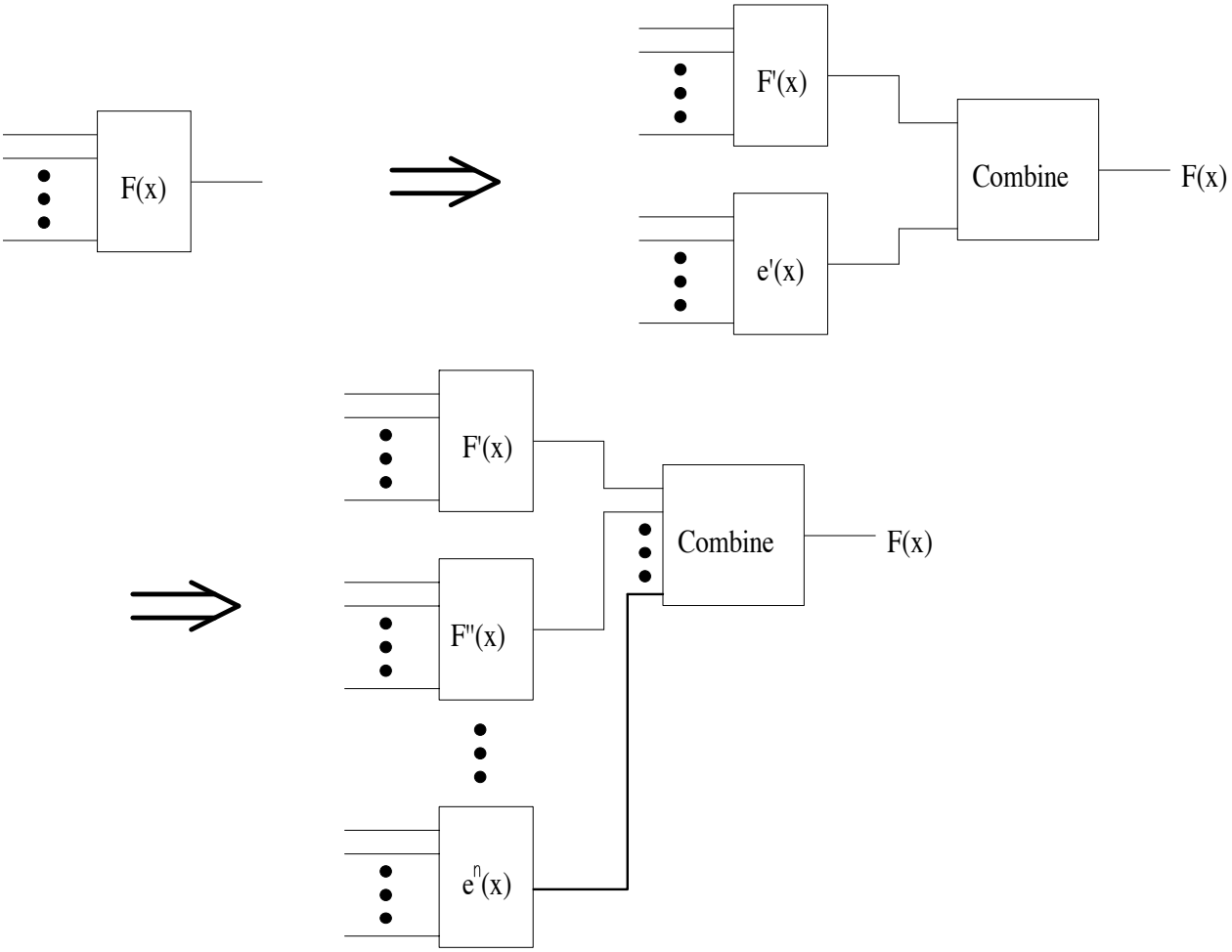

Figure 2: Diagram of Two-Level Synthesis Technique

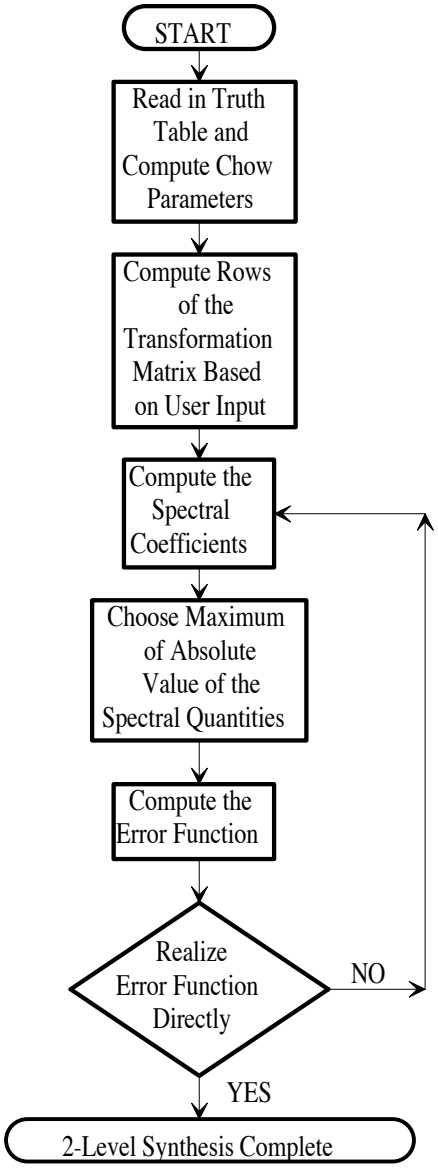

Figure 1: Flowchart of Two-Level Synthesis Technique

[TD92] Damarla, T. Generalized Transforms for Multiple Valued Circuits and their Fault Detection *IEEE Trans. Comp.* **vol. C-41, no. 9** (1992), pp. 1101-1109.

[PP81] Picton, P.D. Realisation of multithreshold threshold logic networks using the Rademacher-Walsh transform, *IEE Proc.* **vol. 128, pt. E, no. 3** (1981), pp. 107-113.

[SH73] Hurst, S.L. The application of Chow parameters and Rademacher-Walsh matrices in the synthesis of binary functions, *Comput. J.* **vol. 16 no. 2** (1973).

[DP86] Pradham, D. K. (editor), **Fault-Tolerant Computing Theory and Techniques Volume I** Englewood Cliffs, NJ.:Prentice-Hall, (1086).

[CE77] Edwards, C.R. The design of easily tested circuits using mapping and spectral techniques, *Radio and Electronic Engineer* **vol. 47, no. 7** (1977), pp. 321-342.

restricting the constituent function operators to a certain type allows the resulting circuit to contain only those operator (gate) types.

If optimization for circuit speed is desired, each spectral coefficient may be weighted by the inverse of the corresponding constituent function delay. This would cause those constituent functions with the least delay and highest correlation to be used.

Existing standard cell logic libraries may also be used with this synthesis technique. All that is required is the output vector of each standard logic cell to be used as a row in the transformation matrix. This allows the synthesis technique presented here to be easily interfaced to existing design environments without the need for changing anything other than the "synthesis engine" itself.

5 Conclusions

We have presented a spectral based synthesis technique for general combinational logic circuits. Although this synthesis technique has been developed within the framework of spectral analysis techniques it may easily be considered to be a method of repeated correlation analysis. The spectral coefficients for a Boolean function have been defined and their various properties have been derived. An iterative algorithm for the synthesis of combinational circuits has been developed and illustrated with examples. The algorithm has been proven to converge when the transformation matrix is of full rank. The proposed method can be custom-tuned to produce circuits with desired properties such as gate count minimization, type of gate, circuit delay minimization, and number of inputs per gate. This result is significant since it allows for complete exploitation of the XOR gate as well as other gate types without resorting to symbolic manipulation of Boolean algebraic equations. Finally, some guidelines for the implementation of the logic circuit synthesis technique have been provided.

References

[AL80] Lloyd, A.M. Design of multiplexor universal-logic-module networks using spectral techniques, *IEE Proc.* **vol. 127, pt. E, no. 1** (1980), pp. 31-36.

[AL79] Lloyd, A.M. A consideration of orthogonal matrices, other than the Rademacher-Walsh types, for the synthesis of digital networks, *J. Electronics*, **vol. 47, no. 3** (1979), pp. 205-212.

[CE75] Edwards, C.R. The Application of the Rademacher-Walsh Transform to Boolean Function Classification and Threshold Logic Synthesis, *IEEE Trans. Comp.*, (1975) pp.48-62.

Table 3: Truth Table Contents of the Function and Error Function

Next, the spectrum for $e(x)$ is computed, resulting in the following vector of spectral coefficients:

$$\underline{S_e^T}[\{F_c(x)\}] = [2, 2, -2, -4, -2, -2, -6] \tag{7}$$

The constituent function corresponding to -6 is chosen since it is a maximum (in magnitude) which is $F_c(x) = x_2 + x_3$. There is a discrepancy in only one place in the truth table, hence we can use a NOR to directly realize this term. The resulting circuit is given in Figure 6.

Note that the only real difference between the two-level and multi-level synthesis techniques is in the choice of new error operators at each iteration. This allows for greater flexibility when it is desired to use one gate type as much as possible since that type may be able to be used for the error operator in each iteration.

4 Implementation Issues of the Synthesis Method

Repeated vector-matrix multiplications can be very time consuming and inefficient. While this is a convenient way to analyze and understand spectral techniques for boolean function synthesis, it is not necessarily the most efficient way to implement these techniques. In particular, the computation of the Chow parameters may easily be performed by accumulating a single sum for each input of the function as the truth table is read in from a file. The sum for each input in the truth table would accumulate the number of times each variable matched (or mismatched) the function output. This accumulated value is N_m (or N_{mm}) for each respective input. The Chow parameters may simply be obtained by using equation 1 to compute the spectral quantity corresponding to each input that N_m was computed for after the entire truth table has been input. If the number of mismatches is a maximum (i.e., $N_{mm} > N_m$) the spectral quantity has a negative sign, otherwise it is positive. This technique allows the Chow parameters to be computed with only $O(n)$ additions and no multiplications (since during the computation of $S_F[F_c(x)]$, the multiplication by 2 can be implemented as a simple bit shift) versus the $O(n)$ multiplications and additions required by naively using the vector-matrix equations.

In general, all spectral coefficients may be computed efficiently as a simple accumulation of sums of the results of comparisons with transformation matrix rows and the current output vector being transformed.

In addition, it is quite easy to add extensions to this technique to produce output with desired properties. For instance, if only two-input gates are desired, all constituent functions in the transformation matrix are restricted to functions of two-inputs only. Also, by

x_1	x_2	x_3	$F(x)$	$F'(x)$	$e(x)$
1	1	1	-1	-1	1
1	1	-1	1	1	1
1	-1	1	-1	1	-1
1	-1	-1	-1	-1	1
-1	1	1	1	-1	1
-1	1	-1	-1	-1	1
-1	-1	1	-1	-1	1
-1	-1	-1	1	1	1

Table 2: Truth Table Contents of the Function and Error Function

Since there is only 1 disagreement between $F(x)$ and $F'(x)$, the terminal condition has been reached and the remaining term is realized directly. The complete circuit is given in Figure 5.

The design of a multi-level circuit with the restriction that all gates are two-input and must be OR type gates as much as possible is given for the second example. The OR operator cannot be used to form a functionally complete set of operators for Boolean algebra. Therefore, at least one other type of gate is needed for the synthesis. This example will also realize the function given in Equation 3.

The transformation matrix is computed in terms of the Chow parameters and two input OR expressions only. The following spectrum results:

$$\underline{S_F^T}[\{F_c(x)\}] = [-2, -2, 2, -2, 2, -2, 2] \tag{6}$$

In this case, all spectral coefficients have equal magnitudes. The constituent function $F_c(x) = x_1 + x_2$ is arbitrarily chosen as a starting point for the synthesis. It is also desirable to use an OR operation for the error function operator since this circuit is to be realized with predominately OR-type gates. By examining the truth table it is seen there are 0 to 1 and 1 to 0 discrepancies which restrict the error operator to be of type XOR. Table 3 illustrates the truth table in terms of the function to be realized and the error function.

x_1	x_2	x_3	$F(x)$	$e(x)$
1	1	1	-1	-1
1	1	-1	1	1
1	-1	1	-1	1
1	-1	-1	-1	1
-1	1	1	1	-1
-1	1	-1	-1	1
-1	-1	1	-1	1
-1	-1	-1	1	-1

The transformation matrix is computed as:

$$
\begin{array}{r}
1 \\
x_1 \\
x_2 \\
x_3 \\
x_1 \oplus x_2 \\
x_1 \oplus x_3 \\
x_2 \oplus x_3 \\
x_1 \oplus x_2 \oplus x_3 \\
x_1 + x_2 \\
x_1 + x_3 \\
x_2 + x_3 \\
x_1 + x_2 + x_3 \\
x_1 x_2 \\
x_1 x_3 \\
x_2 x_3 \\
x_1 x_2 x_3
\end{array}
\left[\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\
1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\
1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\
1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\
1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 \\
1 & -1 & 1 & -1 & -1 & -1 & -1 & -1 \\
1 & -1 & -1 & -1 & 1 & -1 & -1 & -1 \\
1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 \\
1 & 1 & 1 & 1 & 1 & -1 & 1 & -1 \\
1 & 1 & 1 & -1 & 1 & 1 & 1 & -1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & -1
\end{array}\right]
$$

The resulting spectral coefficient vector is:

$$\underline{S_F^T}[\{F_c(x)\}] = [-2, -2, 2, -2, 2, -2, 2, -6, 2, -2, 2, 2, -2, -2, -2, -4] \tag{5}$$

Since the $F_c(x) = x_1 \oplus x_2 \oplus x_3$ has the largest magnitude spectral coefficient, it is chosen and the first portion of the circuit is realized with an exclusive-NOR as shown in Figure 4.

The error function is computed with respect to an exclusive-OR operator since it is the most robust in terms of the possible operators available for providing the combining stage in the circuit. This robustness is due the fact that an XOR can be used to change a 0 to 1 error as well as a 1 to 0 error. The following list describes the properties that determine which gate type may be used as an error operator.

1. XOR : $x \oplus 1 = \overline{x}$, errors may be 1→0 or 0→1
2. AND : $x1 = x$ and $x0 = 0$, all errors must be 1→0
3. OR : $x + 1 = 1$ and $x + 0 = x$, all errors must be 0→1

Returning to the synthesis example, the following table is computed:

constraints that only two-input gates should be used and that the gate type should be OR as much as possible.

Consider the realization of the following function:

$$F(x) = \overline{x}_1\overline{x}_3 + x_1\overline{x}_2x_3 + \overline{x}_1x_2 + x_2\overline{x}_3 \tag{3}$$

First, the function truth table is computed. Table 1 shows the contents of the computed truth table.

x_1	x_2	x_3	F
1	1	1	-1
1	1	-1	1
1	-1	1	-1
1	-1	-1	-1
-1	1	1	1
-1	1	-1	-1
-1	-1	1	-1
-1	-1	-1	1

Table 1: Truth Table Contents of the Function to be Synthesized

Next, the transformation matrix is computed in terms of OR, AND, and XOR operations since we have no constraints on gate types. Also, for each gate type we compute $F_c(x)$ functions for all possible combinations of two or more inputs since we have no restrictions on the maximum number of inputs per gate. Although it is readily apparent that this matrix grows in size with respect to the size of the x vector, it should be pointed out that the matrix space complexity is $O(n^2)$ if the design is restricted to two-input gates of type OR, AND, and XOR. This is easily seen by considering the total number of matrix rows. There will be 3 operator types over all combinations of any two out of n inputs. The total number is:

$$3\begin{pmatrix} n \\ 2 \end{pmatrix} = \frac{3}{2}(n)(n-1) = \frac{3}{2}n^2 - \frac{3}{2}n \tag{4}$$

Added to this value is $n+1$ additional matrix rows for the computation of the Chow parameters [SH73] yielding a total number of rows equal to $3/2n^2 - 1/2n + 1 = O(n^2)$ in size complexity of the matrix.

We always include rows in our transformation matrix that correspond to each component of the x vector and the $F_c(x) = 1$ function (i.e., the Chow parameters) in case of high correlation with regard to a single input to $F(x)$. The spectral coefficient $S_F(1)$ indicates the overall dominant operator of the function. This information is heavily exploited in an alternative spectral based synthesis algorithm that we are working on.

This technique can be used to generate two-level and multi-level tree-type circuits. For two-level realizations, each chosen $F_c(x)$ can be realized in the first stage of the circuit with one multi-input logic gate. The second stage consists of a single combination gate that uses the outputs of all of the chosen constituent functions as its' inputs. The circuits resulting from this synthesis technique are completely fan-out free (CFOF) and have the desirable property of requiring a set of test vectors equal to the number of circuit inputs to test all possible single stuck-at faults [DP86]. As discussed in [CE77], the use of spectral design techniques for logic synthesis is known for the ability to produce easily tested circuits. The diagram in Figure 2 indicates how the two-level circuit is constructed with each iteration.

If a multi-level circuit is desired, the same design procedure is used but the error function computation is performed slightly differently. The difference is that a new combination gate is used in each iteration. This also allows for changing the operator used to define the error function (i.e., the combination operator) at each iteration. The diagram in Figure 3 illustrates the design procedure for a multi-level circuit.

The following theorem states the properties necessary to ensure the convergence of this synthesis algorithm.

Theorem 1 *Any given boolean function, $F(x)$, may be realized with the proposed synthesis technique if the transformation matrix used for the synthesis is of full rank.*

Proof: This proof is a statement that any N-vector can be produced as a combination of a subset of vectors from the set of vectors that are linearly independent over N-space. Each Boolean function to be realized is viewed as a N-vector with components from the binary field. The synthesis procedure described in the preceding text "chooses" a matrix row in each iteration (each row corresponds to a constituent function) to be "combined" with an appropriate combining operator. This process forms the output vector as a combination of row vectors from the transformation matrix. Hence if the transformation matrix contains at least N rows that span N-space, any function output vector can be realized by a finite number of combined transformation matrix rows. □

Note: This synthesis method can not realize any function using a set of constituent functions that do not form a functionally complete set since the resulting transformation matrix will not be of full rank. For example, a function may not be realized if all constituent functions use only the AND operator.

Now some examples of this synthesis technique are given. In the first example, we will assume that there are no restrictions on the number of inputs per logic gate and on the type of logic gate employed. The first example will show how a two-level circuit can be synthesized. The second example shows how a multi-level circuit can be realized with the

□

These results are used to develop logic circuit synthesis methods for combinational circuits. The following sections describe the iterative synthesis method that lends easily to automation and allows for the resulting circuit realization to be customized with respect to the maximum number of inputs per logic gate and the type of logic gate to be used.

3 Synthesis Method

The synthesis method that we propose is dependent on the use of error functions in an iterative fashion. Figure 1 illustrates the overall structure of the synthesis methodology. In fact, we have identified three ways that the use of spectral coefficients may be applied to the combinational logic synthesis problem. This paper will only discuss one of these methods; later papers will present results from the other two methods.

User supplied input consists of the truth table of the function, $F(x)$, to be synthesized, and optionally, the maximum number of inputs per gate, N_{inp}, and preferences of the types of gates, $\{G_t\}$, to be used. The two optional parameters, N_{inp} and the set $\{G_t\}$ are used to determine the set of constituent functions, $\{F_c(x)\}$ to be used in the formation of the transformation matrix. For the sake of generality, we will first assume that the optional parameters are not supplied, however, we will use them in the subsequent synthesis example.

The following list gives a detailed description of each synthesis step for the process depicted in Figure 1.

1. Convert the input truth table to 1's and -1's using a 1 to denote a logic"0" and a -1 to denote a logic "1".

2. Compute the transformation matrix using the constituent functions.

3. Compute the spectral coefficients via vector-matrix multiplication between the transformation matrix and the output vector of the function to be synthesized.

4. Choose the largest (in magnitude) spectral coefficient.

5. Realize the function $F_c(x)$ that corresponds to the chosen coefficient in step 4.

6. Compute the error function, $e(x) = F_c(x) * F(x)$ with respect to some operator, $*$.

7. If $e(x)$ indicates that there are w or fewer errors, go to step 8. Otherwise iterate on the synthesis by going to step 3 and use $e(x)$ as the next function to be synthesized.

8. Combine all the intermediate realizations of the various chosen $F_c(x)$ functions using the $*$ operator and directly realize the function $e(x)$ for the remaining w or fewer errors.

$$S_F[F_c(x)] = 2^n - 2N_{mm} = 2N_m - 2^n \tag{1}$$

Where x is a n-order vector of binary valued inputs to both $F(x)$ and $F_c(x)$.

Proof: The maximum possible absolute value of a spectral coefficient occurs when a row of the matrix is equal to the function output vector or when each component of the vector is the negative of the corresponding entry in the transform matrix row. Hence, the maximum possible absolute value of the spectral coefficient is $|S_F[F_c(x)]| = 2^n$ indicating 100% positive or negative correlation between $F(x)$ and $F_c(x)$. Indeed in this case, either $F(x) = F_c(x)$ or $F(x) = \overline{F_c(x)}$. Each mismatch present in the function output vector and the corresponding matrix row entry always produces a product value of -1. Therefore, N_{mm} mismatches result in a negative partial sum of $-N_{mm}$. The only other possibility is a match which is the complement of mismatches and always produces a product value of $+1$. Since the spectral coefficient for and $F(x)$ and $F_c(x)$ is the difference between the number of matches, N_m, and the number of mismatches, N_{mm}:

$$\begin{aligned} S_F[F_c(x)] &= N_m - N_{mm} \\ &= N_m - [2^n - N_m] \\ &= 2N_m - 2^n \end{aligned}$$

Likewise, substituting N_{mm}:

$$\begin{aligned} S_F[F_c(x)] &= N_m - N_{mm} \\ &= [2^n - N_{mm}] - N_{mm} \\ &= 2^n - 2N_{mm} \end{aligned}$$

Hence, $S_F[F_c(x)] = 2^n - 2N_{mm} = 2N_m - 2^n$. □

Lemma 2 *The following property of spectral coefficients holds:*

$$S_F[F_c(x)] = -S_F[\overline{F_c(x)}] \tag{2}$$

Proof: Let the number of mismatches between the inverse of the constituent function, $\overline{F_c(x)}$, be denoted by N'_{mm} and the corresponding matches denoted by N'_m.

Thus, $N'_m = N_{mm}$. Using this fact and the results from lemma 1:,

$$\begin{aligned} S_F[F_c(x)] &= 2^n - 2N_{mm} \\ &= 2^n - 2N'_m \\ &= -(2N'_m - 2^n) \\ &= -S_F[\overline{F_c(x)}] \end{aligned}$$

The notion of an output vector of a boolean function was described earlier. A qualitative discussion of the derivation of other transform matrices extends this notion such that the rows of the transformation matrix are viewed as output vectors of the constituent functions. The constituent functions are generally simple functions using only a single operator (although they need not be). For instance, if it is desired to compute the correlation between a constituent function, $x * y$, and a function to be transformed, one row of the transformation matrix would consist of the output vector of the constituent function. Hence, the spectral coefficient resulting from the dot product of this row and the function output vector provides a measure of correlation between the overall function and the constituent function, $x * y$. In fact, a measure of correlation with any arbitrary constituent function may be computed in this manner. Each correlation measure (or spectral coefficient) contains the information of the exact number of matching outputs between the constituent function and the transformed function for a given common set of inputs. Before giving a precise relationship between the spectral coefficients and the number of matching outputs, the following notation is defined.

1. n - the number of input variables of a Boolean function
2. N - all possible combinations of the input variables of the binary-valued Boolean function, $N = 2^n$
2. N_m - the number of matches between the outputs of the constituent function and the function to be synthesized for the same input values
3. N_{mm} - the number of mismatches between the outputs of the constituent function and the function to be synthesized
4. $S_F[F_c(x)]$ - the spectral coefficient of $F(x)$ that corresponds to the constituent function, $F_c(x)$
5. $S_F(1)$ - the spectral coefficient of $F(x)$ corresponding to $F_c(x) = 1$
6. $\{F_c(x)\}$ - a set of constituent functions
7. $\underline{S_F}[\{F_c(x)\}]$ - a vector of spectral values computed over the set of constituent functions, $\{F_c(x)\}$

Some properties of the spectral coefficients are derived in the following and they will be used for the development of the synthesis technique given in this paper.

Lemma 1 *For a given function $F(x)$ and a given constituent function $F_c(x)$ the resulting spectral coefficient is given by:*

2 Spectrum of a Boolean Function

First, the following terms are defined:

- **Output Vector of a Boolean Function:** A concatenation of all possible outputs of the function where all "0" values have been changed to a "1" and all "1" values have been changed to "-1".
- **Spectrum of a Boolean function:** A linear transformation of the output vector of the function. This transformed quantity is the "spectral vector" of the transformed function. The components of the the spectral vector are referred to as "spectral coefficients".
- **Transformation Matrix:** The matrix used to perform the linear mapping of a functions' output vector to its spectral vector.
- **Constituent Functions:** Boolean functions whose output vectors are used as the rows of the transformation matrix.

These transformations are linear and they are conveniently visualized as vector-matrix products although the implementation may not necessarily be realized as a series of vector-matrix products. Multiplication and addition operations are performed over the field of integer numbers, not over the binary field. Using "1" and "-1" instead of "0" and "1" for the binary valued digits allows the zero-valued function outputs to accumulate in each spectral quantity.

Each spectral coefficient in the transformed output vector provides a measure of correlation between a particular constituent function and the original function, $F(x)$. This is the underlying principle that is exploited in the synthesis procedure described in this paper.

In general, any set of constituent functions may be used for this technique as long as they form a functionally complete set of operators for Boolean algebra. A transform must be constructed from an orthogonal set of basis functions for the spectrum and original function to form a unique pair. Fortunately, our technique does not rely on transform pair uniqueness. Thus, we make use of transform matrices that are not necessarily orthogonal. The transformation matrix must be full rank however, for convergence of this algorithm to be guaranteed. It has been shown that a transform using the Exclusive-OR operator as the primitive operation for the constituent functions results in the Rademacher-Walsh transform matrix. The Rademacher-Walsh transform matrix is orthogonal with well known properties [CE75]. In this paper, we construct transforms using the OR and AND operators in addition to the XOR to form the constituent functions. The resulting transforms are not necessarily orthogonal.

are used as the basis of this technique are noted and interpreted in this work. In describing these spectral coefficient properties, both a qualitative and quantitative analysis of the spectral coefficients is provided. The results are then used to develop the synthesis technique. Combinational logic synthesis examples are given and a proof of convergence for the proposed synthesis algorithm is provided. In the examples given in this paper, the design method is used for both two-level and multi-level circuits. Various implementation issues of the synthesis technique are discussed and appropriate suggestions are provided. The use of specific design parameters such as the desired gate type and maximum number of inputs per gate are considered. Computer program implementation considerations such as storage and computation requirements are discussed along with methods to minimize the usage of these resources.

Typically, designers generate Boolean expressions at some point in the logic circuit design process regardless of the technique used. In fact, many logic design techniques are considered to be accomplished upon the realization of a Boolean function that is minimized in some sense. In reality, circuits are built from a schematic diagram or some other form of hardware description such as a wire-list or standard CAD tool file format. The synthesis technique discussed in this report does not require an intermediate Boolean function to be realized. Since there is no need for any symbolic manipulation, this technique is especially well suited for implementation as a computer program. Furthermore, the XOR gate is fully exploited as a potential candidate for incorporation into the design in addition to the AND and OR gates. Most design techniques require symbolic manipulation of Boolean functions to achieve inclusion of the XOR gate. Indeed, it has been stated that algebraic techniques in general are not well suited for circuit realization using XOR circuits. The algebraic methods presented here are rigorously developed and are very well suited for XOR type circuits.

The rest of the paper is organized as follows. Section 2 of this paper defines and describes the concept of the spectrum of a Boolean function. In this section, spectral properties that are important for the development and motivation for the synthesis technique are given. The synthesis method is developed and two examples are given in section 3. The first example shows how a two-level logic circuit using multi-input gates may be realized using our synthesis method. The second example illustrates how the synthesis technique can be applied to a circuit that is restricted to being composed of two-input OR gates as much as possible. The second example also shows how this technique is applied for the realization of multi-level circuits. After the examples are given, a discussion of the implementation issues for the logic circuit synthesis methodology is included in section 4. Finally, the conclusions from this work are presented in section 5.

Iterative Combinational Logic Synthesis Techniques Using Spectral Data

Mitchell Aaron Thornton, V. S. S. Nair
Southern Methodist University
Department of Computer Science and Engineering
Dallas, Texas 75275

Abstract

The spectral information of a Boolean function yields data regarding the correlation between the input variables and the output of the function. This paper introduces a spectral based methodology for combinational logic synthesis using linear transforms. An analysis of the properties of the spectra obtained from these transforms is provided and a synthesis algorithm using spectral techniques is presented. This result is significant since it provides an algebraic method for including the XOR gate in the synthesis process without resorting to manipulation of symbolic Boolean equations.

1 Introduction

The design and development of digital systems is becoming increasingly complex and automated. With the evolution of greater amounts of circuitry per single chip comes a need for greater optimization and efficiency in the design process. Spectral based design techniques offer the advantage of providing for the design of a circuit of minimal size for a given functionality. These techniques are highly methodical and are ideal for incorporation into an automated environment. One example of such an environment is the automatic synthesis of a circuit using a hardware description language (HDL). In the past literature, there have been a significant amount of results in the use of various spectral techniques for both binary-valued logic [AL80] [AL79] and multi-valued, or threshold logic [CE75] [TD92] [PP81]. Our approach differs from the previous works in that we allow the synthesized circuit to be composed of any of several types of sub-functions (i.e. AND, OR, combinations of AND/OR), whereas the previous work focused on the realization of logic circuitry with predominately a single type of circuit as the basic building block. In addition, we present the basis for a computer implementation of this procedure and prove its convergence. It has been shown that the Rademacher-Walsh transform provides Boolean function output correlation measures with respect to various combinations of input variables added together via modulo-2 arithmetic [SH73]. This paper builds upon the fundamentals of spectral techniques and describes how they may be applied to the synthesis of combinational Boolean logic circuits.

The particular spectral transforms to be used in this method are introduced and examples are given. Also, certain properties of interest of the resulting spectral coefficients that