Predicting the Functional Complexity of Combinational Circuits by Symbolic Spectral Analysis of Boolean Functions

Enrico Macii Massimo Poncino

Politecnico di Torino
Dip. di Automatica e Informatica
Torino. ITALY 10129

Abstract

Spectral analysis of Boolean functions represents an elegant approach to the problem of estimating the complexity of digital designs. In general, however, the use of spectral techniques in logic synthesis has been very limited in the past because Boolean functions associated with circuits of interest are usually too large and complex to allow the computation of their corresponding spectral representations using traditional techniques. In this paper we present a symbolic formulation of the logic complexity prediction problem, and we propose an ADD-based algorithm that performs well, in terms of both execution time and accuracy in the estimation, on circuits that are sensibly larger than the ones usually handled by the tools currently available. Experimental results are discussed in detail to support this claim.

1 Introduction

The problem of predicting the complexity of a circuit produced by a logic synthesis tool is of primary importance in the process of automatically synthesizing the functional specification of a digital design, and different approaches to its solution have been developed in the last few years. Most of the proposed methods rely on abstract complexity measures based on analytical processing of the Boolean functions describing the circuit being synthesized [1, 2]. An alternative approach, based on information theory, consists of computing the entropy associated to the circuit, which is related to its complexity [3]. In any case, determining such measures is usually computationally very expensive and extremely memory consuming, so that their evaluation becomes infeasible when the sizes of the functions increase over a certain threshold.

For this reason, in [4], the use of spectral transforms has been proposed to re-formulate the logic complexity evaluation problem into a domain where Boolean functions can be handled with considerably less difficulties. Even in this case, however, the sizes of the Boolean functions to be transformed remain a problem; in fact, computing the spectrum of a N-variable Boolean function, specified as a truth table or a set of equations, is a complex operation which involves either the multiplication of the output column(s) of the truth table with a non-singular matrix of size $2^N \times 2^N$ or an analytical computation. Therefore, in the general case, determining the spectral representation of function $f(x_1, \ldots, x_N)$ requires $N \cdot 2^N$ arithmetic operations (additions/subtractions) and 2^N memory locations, when traditional techniques based on explicit minterm enumeration are used; as a consequence, methods as the one presented in [4] have been of limited applicability to real cases.

With the advent of symbolic techniques for the manipulation of Boolean and pseudo-Boolean functions based on decision diagrams, however, spectral transformations have assumed larger relevance. (A pseudo-Boolean function is a function which maps the set $\{0,1\}^N$ onto the set of real numbers.)

Binary Decision Diagrams (BDDs) [5, 6] are a very compact data structure for the representation and manipulation of large Boolean functions, and Algebraic Decision Diagrams (ADDs) [7] are an extension of BDDs to the case of pseudo-Boolean functions. In particular, ADDs provide the user with a complete framework (data structure and operations) for symbolically handling matrices and graphs.

Recently, symbolic algorithms for spectral analysis of Boolean functions have been developed for applications as combinational technology mapping [8], function decomposition for FPGA synthesis [9], and evaluation of Boolean function correlation [10] in the context of state space decomposition algorithms for approximate finite state machine traversal [11, 12].

In this paper we propose an ADD-based algorithm which is able to predict the complexity of a combinational circuit, starting from the functional specification, i.e., the logic equations, of the design being synthesized. In particular, the purpose of this work is two-fold. First, we are interested in showing that, by using symbolic techniques, the spectral approach to logic complexity evaluation can be successfully applied to reasonably large circuits. Second, we aim at proving that the complexity measure we propose here, derived from [13] and different from the one proposed in [14] and employed in [4], is very suitable for logic complexity prediction of circuits generated by state-of-the-art, BDD-based synthesis tools.

Different cost functions can be used to evaluate the logic complexity of a digital design. Literal count and number of product terms in the sum of product realization of the Boolean functions are usually employed for complexity estimation at the specification stage; on the other hand, gate count, transistor count, and chip area are the cost factors considered at the implementation level. Finally, when complexity of a design has to be estimated during the synthesis process, a good indication can be derived from the sizes, i.e., number of nodes, of the BDDs representing the output functions of the circuit under development. We present experimental results that show how the spectral measure we propose in this work can be put in relation to the complexity factors mentioned above.

The rest of this paper is organized as follows. Section 2 reports background information concerning BDDs, ADDs, and spectral analysis of Boolean functions. In Section 3 we give the theoretical foundations of this work; in particular, we introduce criteria for logic complexity estimation in the Boolean domain, we show how they can be re-expressed in the Rademacher-Walsh spectral domain, and we propose an ADD-based algorithm for the computation of such complexity measures; furthermore, we present a criterion to determine the functional complexity of a multiple output combinational circuit. In Section 4 we show different sets of experimental results and we comment on them. Finally, Section 5 is devoted to conclusions and directions for future work.

2 Background

2.1 Binary Decision Diagrams (BDDs)

A BDD is a graph representation of a logic function. Under some conditions, the BDDs are canonical, that is, there is one unique BDD for a given logic function. For example, let us consider he function $f(x, y, z) = xy + x'z + y'z$. The BDD of f is shown in Figure 1.

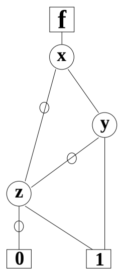


Figure 1: BDD of $f(x, y, z)$.

To obtain the value of the function corresponding to a given assignment of the variables, it suffices to follow a path in the BDD from the root node to a leave by taking the dotted branch when the value of the variable associated with the node is 0, and by taking the dot-free branch otherwise. The value of the leave gives the value of the function.
Sophisticated algorithms exist for efficiently constructing and manipulating BDDs. For a more detailed treatment of this matter, the reader can refer to [5, 6].

2.2 Algebraic Decision Diagrams (ADDs)

Algebraic Decision Diagrams (ADDs) [7] can be viewed as a form of multi-terminal BDDs that support algebraic and arithmetic operations on their terminal nodes. Terminal nodes can hold objects drawn from an arbitrary set, for instance, real numbers. An ADD is a directed acyclic graph representing a set of functions $f_i : \{0,1\}^N \mapsto S$, where S is the carrier of the algebraic structure over which the ADD is defined.
ADDs are particularly suitable for representing matrices. For example, the matrix of Figure 2-a can be represented by the ADD of Figure 2-b. x variables encode row variables, and y variables encode column variables.

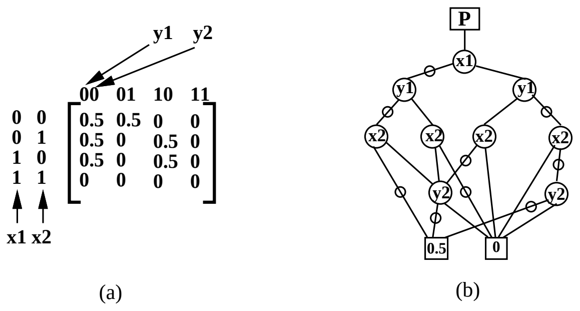


Figure 2: A Matrix and its Corresponding ADD.

Notice that, as in the case of BDDs, the value of the function for a given assignment of the variables is given by the value of the leave reachable from the root of the ADD by taking the dotted branch when the value of the variable associated with the node is 0, and taking the dot-free branch otherwise.
As shown in [15], the use of ADDs has made it possible the realization of algorithms for the manipulation of very large matrices; in fact, the ADD data structure has appeared to be much more memory efficient than traditional storage techniques that exploit structure sparsity.

2.3 Spectral Analysis of Boolean Functions

Let $f : \mathbf{B}^N \mapsto \mathbf{B}$ be a single-output, completely specified Boolean function. (The assumption of f being completely specified is made only because all the Boolean functions we are considering in this paper fall in this category; however, the theory of spectral transforms summarized in the following can be adapted easily to the case of incompletely specified functions.) f can be represented as an integer-valued vector $F = [F_1, \ldots, F_{2^N}]^T$, that is, as a list of the values corresponding to each of the 2^N minterms.
Function f can be mapped onto the transform domain by multiplying its corresponding integer-valued vector F with a $2^N \times 2^N$ non-singular transform matrix.
Among all spectral transforms known in the literature [16], we focus our attention on Rademacher-Walsh transforms; these kinds of transforms have found several applications in the field of logic synthesis. The Rademacher-Walsh spectrum of a given Boolean function, f, written as $\widehat{F}$, is given by:

$$\widehat{F} = W^{(N)} F, \tag{1}$$

where $W^{(N)}$ is the Rademacher-Walsh transform matrix.
Notice that the inverse transformation is possible, that is, F can be recovered from $\widehat{F}$ as follows:

$$F = W^{(N)} \widehat{F}.$$

The encoding used for vector F determines the type of the spectrum, that is, the R or the S spectrum. In the first case, F is encoded by its original values: 0 for false minterms of f, that is, minterms for which f assumes the logic value 0, and 1 for true minterms of f, that is, minterms for which f assumes the logic value 1. In the latter, false minterms are denoted with 1, and true minterms with -1. Only the R spectrum will be considered throughout this paper.
The transform matrix, $W^{(N)}$, can be obtained with a recursive formulation as follows:

$$W^{(N)} = \begin{bmatrix} W^{(N-1)} & W^{(N-1)} \\ W^{(N-1)} & -W^{(N-1)} \end{bmatrix}, \text{ where } W^{(0)} = [1]. \tag{2}$$

As a simple example, let us compute the spectrum of the function f realized by a two-input NAND gate. The integer-valued vector F corresponding to f is:

$$F = [1\ 1\ 1\ 0]^T.$$

To compute $\widehat{F}$, we need to construct the Rademacher-Walsh matrix of order 2, that is:

$$W^{(2)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

Therefore, we obtain $\widehat{F}$ as follows:

$$\widehat{F} = W^{(2)} F = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

Each Rademacher-Walsh spectral coefficient, except for the first one, tells us the minterm-by-minterm resemblance of the function to a linear function (EXOR) of some combination of the input variables corresponding to the binary representation of the coefficients [16]. For example, a large magnitude (absolute value) spectral coefficient at the coordinate 3 (binary 11) indicates that a large part of the function can be represented as $x_1 \oplus x_2$.

3 Predicting Circuit Complexity

3.1 Complexity of a Boolean Function

As mentioned in Section 1, several criteria to estimate the complexity of a digital design have been presented in the past. Particularly interesting is the one proposed in [14]: Given a generic N-input Boolean function, f, a measure of its ***simplicity***, μ, is given by the number of true minterms of f at unit Hamming distance; in formula:

$$\mu(f) = (\sum_{||\tau=1||} (\sum_{x=1}^{2^N} f(x) f(x \oplus \tau))), \qquad (3)$$

where $x = \sum_{i=1}^{N} x_i 2^{i-1}$, $\tau = \sum_{i=1}^{N} \tau_i 2^{i-1}$, and $||\tau||$ is the number of ones in the binary representation of τ, i.e., $(\tau_1, \ldots, \tau_N)$. Notice that function $\sum_{x=1}^{2^N} f(x) f(x \oplus \tau)$ is the ***auto-correlation*** function of f, and it is usually denoted as $B^{(f,f)}(\tau)$. The higher the values of μ, the simpler the corresponding Boolean function. Logic complexity results presented in [4], obtained using the equivalent of Equation 3 in the spectral domain as estimation criterion, were not particularly meaningful, except for a very limited class of arithmetic circuits, i.e., N-bit adders. The poor behavior of the estimator can be explained as follows. Complexity $\mu(f)$ was computed by taking into account only the true minterms of function f; however, the benchmarks on which the estimator was tested were produced by synthesis tools that exploited the information on both the on-set and the off-set of f to optimize and map onto a library the functional specification of the design.

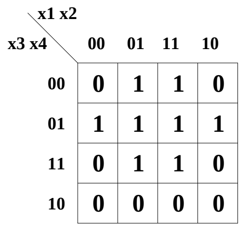
This is the reason why, in the realization of our symbolic logic complexity predictor, we have used a modified version of μ, in the remaining of the paper indicated as χ, that counts the number of adjacent pairs of assignments for which function f takes the same value (0 or 1) for both assignments in the pair. The definition of $\chi(f)$ is the following [13]:

$$\chi(f) = \sum_{||\tau=1||} (B^{(f,f)}(\tau) + B^{(f',f')}(\tau)), \qquad (4)$$

where n, x, τ, and $||\tau||$ are the same as in Equation 3.

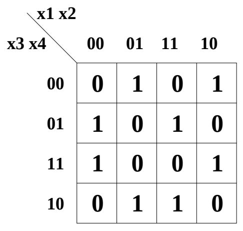
As demonstrated by Karpovsky in [17], measure χ provides a complexity estimation which is more accurate than the one provided by criterion μ, independently on the synthesis algorithms used to produce the final implementation of the design.

As an application example of Equation 4, let us estimate the complexity of functions $f(x_1, x_2, x_3, x_4)$ and $g(x_1, x_2, x_3, x_4)$, whose specification is given in Figure 3.



x3 x4 \ x1 x2	00	01	11	10
00	0	1	1	0
01	1	1	1	1
11	0	1	1	0
10	0	0	0	0

f(x1,x2,x3,x4)



x3 x4 \ x1 x2	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	1	0	0	1
10	0	1	1	0

g(x1,x2,x3,x4)

Figure 3: Karnaugh Maps of Functions f and g.

We have that $\chi(f) = 32$, and $\chi(g) = 16$, so the complexity of f is clearly much smaller than the one of g, as it can be easily guessed by looking at the position of the 1's in the maps of Figure 3.

3.2 Spectral Formulation

Equation 4 gives us a way to estimate the complexity of a logic function, f, through operations in the Boolean domain. As said in Section 1, it may be computationally too expensive performing these operations in such domain; on the other hand, they may result particularly cheap if performed in the transform domain, that is, on the spectral representation $\widehat{F}$ of f.

Hurst *et al.* in [13] have proved that Equation 4 can be re-expressed in such a way that $\chi(f)$ is determined by operating directly on $\widehat{F}$ instead of f. We have:

$$\begin{aligned} \chi(f) &= N \cdot 2^N - \tfrac{1}{2^{N-2}} (\textstyle\sum_{x=1}^{2^N} ||x|| (\widehat{F}_x)^2) \\ &= N \cdot 2^N - \tfrac{1}{2^{N-2}} \textstyle\sum_{x=1}^{2^N} S_{\widehat{F}}(x), \end{aligned} \qquad (5)$$

where $\widehat{F}_x$ is the coefficient in position x of the spectral representation $\widehat{F}$ of f, and $||x||$ is the number of ones in the binary representation of x.

Equation 5 is particularly suitable for a symbolic realization through ADDs, that is, for being computed without explicitly enumerating all the 2^N elements of each $\widehat{F}$, as it will be shown in detail in Section 3.3.

3.3 The Symbolic Algorithm

The reason why spectral techniques have never been used extensively in logic synthesis is because of storage and manipulation problems related to the sizes of both transform matrices and spectral representations of large Boolean functions. Recently, however, the use of ADDs to store integer-valued functions has proved to be extremely effective in handling these kinds of functions in the transform domain (see, for example, the work of Clarke *et al.* [8], where Rademacher-Walsh spectral representations of Boolean functions with up to 300 input variables have been used for technology mapping of combinational circuits). Though in the following we use symbolic algorithms to estimate the complexity of Boolean functions represented in the spectral domain, techniques to build transform matrices and spectral representations are not within the scope of this paper and, therefore, they are not treated here; for further detail on these techniques, the reader may refer to [7, 18].

The pseudo-code of procedure Predict_Complexity for the evaluation of the $\chi(f)$ (see Equation 5) is shown in Figure 4.

```
procedure Predict_Complexity ($f$, $N$) {
1    $W^{(N)}(x,y)$ = AddWalsh($N$);
2    $\widehat{F}(x)$ = AddMatrixMult($W^{(N)}(x,y)$, $F(y)$);
3    $\widehat{F}^2(x)$ = AddApply($\widehat{F}(x)$, $\widehat{F}(x)$, $*$);
4    $S_{\widehat{F}}(x)$ = AddApply($\widehat{F}^2(x)$, Ones_Count($x$,$N$), $*$);
5    $\widetilde{\chi}(f) = \backslash_x^+ S_F(x)$;
6    $\chi(f) = N \cdot 2^N - \frac{1}{2^{N-2}} \widetilde{\chi}(f)$;
7    return($\chi(f)$);
}

procedure Ones_Count ($x$,$N$) {
    foreach ($x_i \in x$) {
        $Res(x)$ = AddApply($Res$, $x_i$, $+$);
    return ($Res(x)$);
}
```

Figure 4: The Predict_Complexity Algorithm.

In order to emphasize the symbolic nature of the algorithm, we represent matrices as functions of (x, y), and vectors as functions of x or y, where $x = (x_1, x_2, \ldots)$ is the set of row variables and $y = (y_1, y_2, \ldots)$ is the set of column variables.

The procedure receives, as input parameters, the ADD representation of a Boolean function, f, and the number N of the circuit inputs, and returns the value of its logic complexity, $\chi(f)$. In Line 1, we build the transform matrix of order N, $W^{(N)}$, by using procedure `AddWalsh`, and we determine the square $\widehat{F}^2$ of the spectral representation of f (Lines 2 and 3 of the pseudocode). In Line 4 we calculate $S_{\widehat{F}}(x) = ||x||(\widehat{F}_x)^2$, and in Lines 5 and 6 we compute $\chi(f)$ through existential abstraction of the x variables from $S_{\widehat{F}}(x)$ and normalization. Finally, in Line 7 we return $\chi(f)$. Notice that procedure `Ones_Count`, used to compute $S_{\widehat{F}}(x)$, returns the ADD of the N-bit *Tally Function* of number x.
It is important to observe that the algorithm above heavily relies on the performance of matrix multiplication; procedure `AddMatrixMult` operates on two matrices represented as ADDs using a recursive algorithm based on Boole's expansion theorem [19] and top variable splitting; for a detailed discussion on the efficiency of procedure `AddMatrixMult` and a comparison to traditional sparse matrix multiplication algorithms, in the case of multiplication of very large matrices, the reader may refer to [7, 15].

3.4 Complexity and Support

Comparing the complexity of two functions, f_1 and f_2, as in the example of Figure 3, is meaningful as long as the two functions depend on the same number of input variables. In other words, the measure of simplicity is relative to the number of input variables.
To clarify this, let us consider the following example. Given functions

$$f_1 = x_1', \quad \text{and} \quad f_2 = x_1'x_2' + x_3(x_1' + x_2'),$$

let us compute the corresponding values of χ. Applying Equation 4, we obtain $\chi(f_1) = 0$, and $\chi(f_2) = 12$. This result is clearly in contrast with the expected interpretation that higher values of χ imply "simpler" functions. f_2 is indeed a more complex function than f_1. Despite this fact , the result we have computed is correct; f_1 is more "complex" than f_2 with respect to its support size. In particular, f_1 is the "most" complex one-variable function, (which is indicated by $\chi(f_1)$ being 0), while f_2 is relatively complex with respect to three-variable functions. This example suggests that, in order to get a meaningful complexity measure, it is necessary to compare values of χ computed with respect to the same variable support. If we re-compute $\chi(f_1)$ regarding f_1 as a function of three variables (i.e., with the same support as f_2), we get $\chi(f_1) = 16$, that now tells us that f_1 is a simpler three-variable function than f_2.

3.5 Circuit Complexity

In this section we describe how we can use the previous definitions to evaluate the complexity of a combinational logic circuit. In the general case of a multiple output combinational circuit realizing function $f = (f_1, \ldots, f_m)$, we are interested in computing a global complexity measure from the individual values of $\chi(f_i)$. A simple way to obtain this measure is to average the values of $\chi(f_i)$ over all circuit primary outputs. In formula, we have:

$$\chi_C(f) = \frac{\sum_{i=1}^{m} \chi(f_i)}{m}, \qquad (6)$$

where m is the number of circuit outputs.
Notice that, as shown in Section 3.4, $\chi(f_i)$ has to be computed with respect to the total circuit support, that is, the set of all the primary inputs of the circuit.

For a N-variable function, the values of χ_C range from 0 to $N \cdot 2^N$. When we have to compare different circuits with a different number of inputs, it is convenient to normalize χ_C with respect to its maximum value. We obtain:

$$\chi_C'(f) \quad = \quad \frac{\chi_C(f)}{N \cdot 2^N}, \qquad (7)$$

where N is the number of primary inputs of the circuit.

4 Experimental Results

In this section we present complexity estimation results we have obtained on some circuits taken from both the MCNC'91 [20] and the ISCAS'89 [21] benchmark suites. All the experiments have been run on a DEC-Station 5000/240 with 64M of memory. Circuit optimization has been performed using SIS [22], and technology mapping to obtain area information has been done using a commercial cell library (see [23] for details). Notice that, for sequential benchmarks, only the combinational portion of logic has been considered, that is, state inputs and state outputs have been treated as primary inputs and primary outputs, respectively.
The factor χ_C' is an absolute measure of the logic complexity of a design. To show this experimentally, for each benchmark we have considered two different optimizations, obtained through the SIS scripts `algebraic` and `rugged`, and we have compared their complexities. As expected, the value of χ_C' was constant for all the different realizations, while other measures, such as average number of ADD nodes, total literal count, and area presented noticeable differences. While for literal count and area this fact is very intuitive, the reason why the number of average ADD nodes does not remain constant is quite subtle. In fact, given that χ_C' is computed from the ADDs of the circuit, one would expect a direct relation between the number of nodes and the value of χ_C'. However, ADD sizes are sensitive to their variable orderings, and such orderings are usually computed statically through structural analysis of the circuit implementation; therefore, different optimizations may change the structure of the circuit and, thus, induce a different variable ordering which may influence the total ADD sizes. On the other hand, χ_C' is truly a functional measure, because its value (but, of course, not the time required to compute it) is independent from the ADD variable ordering used during the calculation.
Data for the experiments above are summarized in Table 1. In particular, column *Opt Script* indicates what SIS script has been used for optimization, column χ_C' gives the complexity of the design, columns *Nodes* and *Lit* the average number of ADD nodes and the number of literals of all the output functions, and column *Area* the area, in μm^2, of the mapped circuit. Finally, column *CPU* presents the CPU time required to determine χ_C'.
As a general comment, notice that the values of χ_C' are sensibly biased towards the higher end of the range. This is because most circuits have many output functions that only depend on a subset of the circuit inputs; as a consequence, these functions usually contribute values to the average computed in Equation 6 which are close to $N \cdot 2^N$.
In Table 2 we present results for another set of experiments. We have considered pairs of circuits that are known to be functionally equivalent, but with different logic structure, and we have measured their logic complexities. For all the pairs, the value of χ_C' has shown to be the same, as reported in column χ_C' of the table. However, other complexity measures (total literal count and area), which are realization dependent, have not been able to catch the information about circuit equivalence.
As final comment, we would like to stress the point that our complexity prediction algorithm can deal with circuits which are much larger than the ones tools currently available can handle (see, for example, results in [4]).

Circuit	*Inputs*	*Outputs*	*Opt Script*	$\chi'_C(f)$	*Nodes*	*Lit*	*Area*	*CPU*
pcle	19	9	none	0.932	10.6	71	69136	0.3
			algebraic	0.932	10.6	78	69136	0.4
			rugged	0.932	10.6	69	67280	0.5
cordic	23	2	none	0.958	41.0	194	98368	0.7
			algebraic	0.958	41.0	83	66352	0.7
			rugged	0.958	41.0	64	55680	0.8
frg1	28	3	none	0.950	68.6	130	116464	33.5
			algebraic	0.950	68.6	146	116000	48.3
			rugged	0.950	68.6	136	123424	72.1
my_adder	33	17	none	0.885	27.8	257	193024	7.9
			algebraic	0.885	27.8	240	179104	10.1
			rugged	0.885	27.8	192	165648	8.0
s208	19	10	none	0.965	10.1	166	84448	0.1
			algebraic	0.965	11.5	101	84448	0.2
			rugged	0.965	12.8	77	70528	0.2
s298	17	20	none	0.924	7.4	244	138736	0.1
			algebraic	0.924	9.4	138	139200	0.2
			rugged	0.924	8.2	116	135952	0.2
s344	24	26	none	0.944	6.6	269	176784	0.3
			algebraic	0.944	6.8	176	163792	0.3
			rugged	0.944	7.6	149	153584	0.4
s510	25	26	none	0.929	21.1	424	315056	1.4
			algebraic	0.929	24.6	303	308560	2.6
			rugged	0.929	23.3	242	285360	3.2
s820	23	24	none	0.973	19.1	757	464000	3.0
			algebraic	0.973	19.1	359	330832	6.3
			rugged	0.973	21.0	301	322016	8.4
s1196	32	32	none	0.964	59.5	1009	679296	83.3
			algebraic	0.964	73.7	675	634288	125.2
			rugged	0.964	65.0	574	619440	66.6

Table 1: Complexity Results for Differently Optimized Circuits.

Circuit	*Inputs*	*Outputs*	$\chi'_C(f)$	*Lit*	*Area*	*CPU*
s344	24	26	0.944	269	176784	0.3
s349	24	26	0.944	273	180960	0.3
s526	24	27	0.946	445	270048	0.4
s526n	24	27	0.946	445	270512	0.3
s820	23	24	0.973	757	464000	3.2
s832	23	24	0.973	769	476992	3.1
s1196	32	32	0.964	1009	679296	83.7
s1238	32	32	0.964	1041	756320	83.5

Table 2: Complexity Results for Functionally Equivalent Circuits.

5 Conclusions and Future Work

Although the use of spectral transforms for estimating the complexity of a digital design has proved to be particularly appealing, its applicability has been very limited in the past. The main reason for this has to do with the difficulty of constructing the spectral image of large Boolean functions. In the last few years, however, the use of symbolic methods for Boolean and pseudo-Boolean function manipulation has made it possible to successfully use spectral techniques for technology mapping, FPGA-based synthesis, logic decomposition, and some other applications.

In this paper we have discussed criteria for predicting the logic complexity of a digital circuit given its functional specification in terms of algebraic equations. Then, we have shown how such criteria can be more easily re-expressed in the Rademacher-Walsh spectral domain. The measures of above determine the logic complexity of a design by examining the component functions on a minterm-by-minterm basis; this imply the non-applicability of these techniques to circuits whose associated Boolean functions are large. To overcome this problem, we have proposed an ADD-based algorithm for evaluating the complexity of a Boolean function that does not require to explicitly enumerate all its minterms; instead, the complexity measure is determined by considering all the minterms at the same time, that is, using implicit enumeration. As a consequence, circuits we have been able to analyze are much larger than the ones tools similar to ours are able to handle. Besides the ability of dealing with larger circuits, another issue that we have taken into account while realizing our procedures is the accuracy in the logic complexity estimation; more specifically, we have chosen, as complexity criterion, a measure which is particularly adequate to analyze circuits whose implementation is obtained through modern, BDD-based logic synthesis tools. Different sets of experimental results have been reported on benchmark circuits in order to show the effectiveness of our algorithms.

Spectral coefficients can be used to test the functional behavior of a combinational network. The values of certain coefficients are empirically measured, and the results are compared to the expected values. A discrepancy indicates a fault. The coefficients to be tested are called the signature of the network. Studying symbolic algorithms for efficiently selecting a signature for a given circuit and a given class of faults is one of the objectives of our current research. Another area of application of spectral techniques is the synthesis of combinational circuits; we are currently investigating the possibility of applying symbolic computations based on ADDs to algorithms like the ones proposed by Thornton and Nair in [24, 25].

Acknowledgments

We would like to thank professor Fabio Somenzi for valuable suggestions on the realization of ADD-based algorithms.

References

[1] M. G. Karpovsky, "Harmonic Analysis of Over Finite Commutative Groups in Linearization Problems for Systems of Logical Functions," Information and Control, Vol. 33, No. 2, pp. 142-165, February 1977.

[2] C. Moraga, "Comments on a Method of Karpovsky," Information and Control, Vol. 39, No. 3, pp. 243-246, December 1978.

[3] K. T. Cheng, V. D. Agrawal, "An Entropy Measure for the Complexity of Multi-Output Boolean Functions," DAC-27: ACM/IEEE Design Automation Conference, pp. 302–305, Orlando, FL, June 1990.

[4] D. Varma, E. A. Trachtenberg, "On the Estimation of Logic Complexity for Design Automation Applications," ICCD-90: IEEE International Conference on Computer Design, pp. 368-371, Cambridge, MA, September 1990.

[5] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Transactions on Computers, Vol. C-35, No. 8, pp. 79-85, August 1986.

[6] K. S. Brace, R. Rudell, R. Bryant, "Efficient Implementation of a BDD Package," DAC-27: ACM/IEEE Design Automation Conference, pp. 40-45, Orlando, FL, June 1990.

[7] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, "Algebraic Decision Diagrams and their Applications," ICCAD-93: ACM/IEEE International Conference on Computer Aided Design, pp. 188-191, Santa Clara, CA, November 1993.

[8] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, J. Yang, "Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping," DAC-30: ACM/IEEE Design Automation Conference, pp. 54-60, Dallas, TX, June 1993.

[9] Y. T. Lai, M. Pedram, S. Sastry, "BDD-Based Decomposition of Logic Functions with Applications to FPGA Synthesis," DAC-30: ACM/IEEE Design Automation Conference, pp. 642-647, Dallas, TX, June 1993.

[10] E. Macii, M. Poncino, "Using Symbolic Rademacher-Walsh Spectral Transforms to Evaluate the Correlation between Boolean Functions," GLSVLSI-95: IEEE 5th Great Lakes Symposium on VLSI, pp. 112-116, Buffalo, NY, March 1995.

[11] H. Cho, G. D. Hachtel, E. Macii, B. Plessier, F. Somenzi, "Algorithms for Approximate FSM Traversal," DAC-30: ACM/IEEE Design Automation Conference, pp. 25-30, Dallas, TX, June 1993.

[12] H. Cho, G. D. Hachtel, E. Macii, M. Poncino, F. Somenzi, "A Structural Approach to State Space Decomposition for Approximate Reachability Analysis," ICCD-94: IEEE International Conference on Computer Design, pp. 236-239, Cambridge, MA, October 1994.

[13] S. L. Hurst, D. M. Miller, J. C. Muzio, "Spectral Method of Boolean Function Complexity," Electronics Letters, Vol. 18, No. 13, pp. 572-574, June 1982.

[14] D. Varma, E. A. Trachtenberg, "Design Automation Tools for Efficient Implementation of Logic Functions by Decomposition," IEEE Transactions on Computer Aided Design, Vol. CAD-8, No. 8, pp. 901-916, August 1989.

[15] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, Algebraic Decision Diagrams and their Applications, Internal Report, Dept. of Electrical and Computer Engineering, University of Colorado at Boulder, April 1993.

[16] S. L. Hurst, D. M. Miller, J. C. Muzio, Spectral Techniques in Digital Logic. Academic Press Inc., New York, NY, 1985.

[17] M. G. Karpovsky, Finite Orthogonal Series in the Design of Digital Devices, John Wiley and Son, New York, NY, 1977.

[18] E. M. Clarke, M. Fujita, P. C. McGeer, K. L. McMillan, J. Yang. "Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation," IWLS-93: International Workshop on Logic Synthesis, Lake Tahoe, CA, May 1993.

[19] G. Boole, The Mathematical Analysis of Logic, Macmillan, 1847, Reprinted by B. Blackwell, Oxford, UK, 1951

[20] S. Yang, Logic Synthesis and Optimization Benchmarks User Guide, Version 3.0, MCNC, Research Triangle Park, NC, January 1991.

[21] F. Brglez, D. Bryan, K. Koźmiński, "Combinational Profiles of Sequential Benchmark Circuits," ISCAS-89, pp. 1929-1934, Portland, OR, May 1989.

[22] E. M. Sentovich, K. J. Singh, C. W. Moon, H. Savoj, R. K. Brayton, A. Sangiovanni-Vincentelli, "Sequential Circuits Design Using Synthesis and Optimization," ICCD-92: IEEE International Conference on Computer Design, pp. 328-333, Cambridge, MA, October 1992.

[23] R. I. Bahar, G. D. Hachtel, E. Macii, F. Somenzi, "A Symbolic Method to Reduce Power Consumption of Circuits Containing False Paths", ICCAD-94: ACM/IEEE International Conference on Computer Aided Design, pp. 368-371, San Jose, California, November 1994.

[24] M. A. Thornton, V. S. S. Nair, "An Iterative Combinational Logic Synthesis Technique Using Spectral Information," EuroDAC-93: IEEE European Design Automation Conference, pp. 358-363, Hamburg, Germany, September 1993.

[25] M. A. Thornton, V. S. S. Nair, "Parity Function Detection and Realization Using a Small Set of Spectral Coefficients," IWLS-95: ACM/IEEE International Workshop on Logic Synthesis, pp. 8.39-8.47, Lake Tahoe, CA, May 1995.