Synthesis by Spectral Translation Using Boolean Decision Diagrams

Jeffery P. Hansen Masatoshi Sekine
Toshiba ULSI Research Laboratories
1, Komukai Toshiba-cho, Saiwai-ku, Kawasaki 210, Japan

Abstract

Many logic synthesis systems are strongly influenced by the size of the SOP (Sum-of-Products) representation of the function being synthesized. Two-level PLA (Programmable Logic Array) synthesis and many multi-level synthesis systems perform poorly without a good SOP representation of the target function. In this paper, we propose a new spectral-based algorithm using BDDs (Boolean Decision Diagram) to transform the target function into a form that is easier to synthesize by using a linear filter on the inputs. Using the methods described in this paper, we were able to perform spectral translation on circuits with many more inputs and much larger cube sets then previously possible. This can result in a substantial decrease in delay and area for some classes of circuits.

1 Introduction

Spectral methods for design and classification of boolean functions date back even before the first digital computers. The use of spectral methods in circuit design is also quite old and dates back from the early 1960s. While the theory was fascinating, at the time its implementation was impractical for actual circuit design due to the exponential complexity of computing spectral coefficients and the need for an explicit representation of the truth table of the function being transformed. However, recent innovations in implicit function representations [1][2] and in relating these representations to the spectral domain[9], has renewed considerable interest in spectral techniques [4][5][9].

One application for spectral methods is spectral translation [6]. In spectral translation, a function f is divided into a linear block σ and a non-linear block f' (Figure 1) such that $f(X) = f'(\sigma X)$ and where the matrix product σX is evaluated over the Galois Field GF(2). The synthesis problem is to find a linear block σ that minimizes the complexity of f'. The non-linear block f' is typically synthesized with a traditional two-level or multi-level synthesis tool, and the linear block σ is usually implemented as a network of XOR (exclusive OR) gates. While it is difficult to derive σ in the boolean domain, by transforming f to the spectral domain, the synthesis of σ becomes relatively easy. Depending on the function, by adding only a few XOR gates at the inputs the number of terms and literals in the SOP form of f' can be decreased by orders of magnitude compared to the SOP expression for f. This can have a significant impact on the final synthesized circuit.

Unfortunately, the computational complexity of spectral translation methods grows exponentially with the number of inputs making direct approaches impractical. One solution to this problem is to compute the spectral coefficients from the cube set[10] and use heuristics to find the maximum coefficient. Using this approach, larger functions can be handled than by previous brute force approaches, but when the cube set becomes large, this method also becomes impractical.

In this paper, we present a new BDD based logic synthesis tool Spectre (SPECtral TRanslation Engine). Spectre reads a specification file for a multi-output boolean function and outputs its result as a linear block (σ) and a non-linear block(f'). The linear block is a network composed of only XOR gates which translates the input basis to a new basis. The non-linear block is a multi-output boolean function in terms of the translated basis and is synthesized using a standard synthesis package such as SIS (Sequential Interactive System).

By using a BDD-based representation for spectra, we were able to avoid the usual exponential increase in computing cost for the spectral coefficients of functions with large numbers of inputs. We applied our algorithms to the benchmarks from MCNC and other sources and found that the number of literals and terms in the SOP form decreased to some extent for most functions, and by several orders of magnitude for a few functions. For some circuits, we were able to cut both area and delay by almost half.

2 The Walsh Spectrum

Consider an n-input boolean function $f(X)$ where $X = [x_0, x_1, x_2, \ldots, x_{n-1}]^T$. Let X have the binary encoding $x = \sum_{i=0}^{n-1} x_i 2^i$, and let $Y^n(f)$ be the 2^n element vector representing[1] the truth table of f such that y_x is 1 when $f(X)$ is logic-0, and y_x is -1 when $f(X)$ is logic-1. The Walsh spectrum[6] of f is the 2^n element vector:

$$S^n(f) = \sum_{u=0}^{2^n-1} s_u \xi_u = W^n Y^n(f) \quad (1)$$

where ξ_u is the 2^n element basis vector in which the u^{th} element is 1, and all other elements are 0, and where W^n is the $2^n \times 2^n$ Hadamard matrix defined as:

$$W^n = \left[\begin{array}{cc} W^{n-1} & W^{n-1} \\ W^{n-1} & -W^{n-1} \end{array} \right], \qquad W^0 = 1 \quad (2)$$

The Walsh matrix is a symmetric matrix consisting only of 1s and -1s. If we define $u = \sum_{i=0}^{n-1} u_i 2^i$ as a binary encoding similar to the encoding x, then u^{th} row vector of W^n

[1] In a p-valued logic system, typically $e^{\frac{2\pi\sqrt{-1}}{p}\delta}$ is used to represent the logic level δ where $\delta \in \{0, 1, \ldots, p-1\}$.

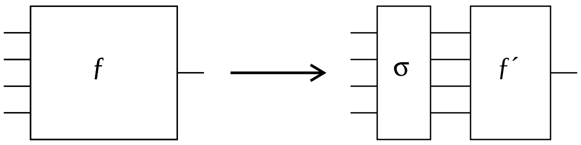

Figure 1: Spectral Translation of a Function

is $W^n_{u*} = Y^n(\bigoplus_{i=0}^{n} x_i u_i)$. That is, as an encoding for the XOR sum of variables x_i for which $u_i = 1$. The rows of W^n have the property that $W^n_{u*} \cdot W^n_{v*}$ is 2^n when $u = v$, and 0 when $u \neq v$ and thus form a set of orthogonal vectors. The spectral coefficient $s_u = W^n_{u*} \cdot Y^n(f)$ is equal to the number of agreements minus the number of disagreements between $Y^n(f)$ and W^n_{u*} and represents the similarity between $Y^n(f)$ and W^n_{u*}. It has a large magnitude when there is a high correlation and a low magnitude when there is a low correlation. A negative s_u indicates a negative correlation. Where convenient, we will also write $s_{x_i x_j \ldots}$ to indicate the spectral coefficient $s_{2^i+2^j+\ldots}$. We will also define the "order" of a spectral coefficient s_u as $||u|| = \sum_{i=0}^{n-1} u_i$ which is equal to the number of variables in the XOR sum for row W^n_{u*} of the Walsh transform matrix.

Spectral invariance operators on a function f are operators for which the position and sign of the coefficients may change, but the set of magnitudes that appear in the spectrum do not change. There are five spectral invariance operations [6], but in this paper we will primarily be concerned with the translation operation $f_{x_i = x_i \oplus x_j}$ in which an input x_i is replaced by that input XORed with another input x_j. This operation causes the following spectral parameters to be exchanged:

$$s_{x_i \alpha} \leftrightarrow s_{x_i x_j \alpha} \tag{3}$$

α represents all combinations of variables other than x_i and x_j. Since spectral invariance operators are reversable, they divide the set of n-input boolean functions into a set of equivalence classes, which are generally far fewer in number compared to the set of all boolean functions. Spectral translation techniques involve the application of these operations to find the "simplest" function in the equivalence class.

Notice that the exchange of spectral coefficients in (3) will cause the order of the exchanged values to either increase or decrease by one. It has been noted [6] when the high order coefficients of a function have high magnitude, the function generally does not have a good AND/OR representation. This suggests that we would like to find translations which move high magnitude coefficients to low order positions, and low magnitude coefficients to high order positions.

3 Normalized Spectral Product Sum

In this section, we introduce the NSPS (Normalized Spectral Product Sum) transformation. The NSPS transform has these advantages: it is easily computed from the BDD; and low-order NSPS coefficients contain information about high-order Walsh coefficients. The NSPS transform of n-input functions f and g is defined in terms of the SPV (Spectral Product Vector) of f and g, which is defined as the 2^n element vector:

$$R^n(f,g) = \sum_{u=0}^{2^n-1} r_u \xi_u = \sum_{u=0}^{2^n-1} (\xi_u \cdot S^n(f))(\xi_u \cdot S^n(g))\xi_u \tag{4}$$

The vector $R^n(f,g)$ is the vector resulting from multiplying the corresponding elements of $S^n(f)$ and $S^n(g)$. The NSPS

S		b	ab	a
	2	6	-2	2
d	2	6	-2	2
cd	2	6	6	-6
c	-6	-2	-2	2

(a)

Q		b	ab	a
	16	20	12	12
d	20	28	20	20
cd	28	18	36	36
c	20	20	20	20

(b)

Figure 2: Spectral Map (a) and NSPS Map (b)

transform of f and g is then the 2^n element vector:

$$Q^n(f,g) = \sum_{u=0}^{2^n-1} q_u \xi_u = K^n R^n(f,g) \tag{5}$$

where K^n is the $2^n \times 2^n$ matrix:

$$K^n = \begin{bmatrix} \frac{1}{2}K^{n-1} & \frac{1}{2}K^{n-1} \\ 0 & K^{n-1} \end{bmatrix}, \qquad K^0 = 1 \tag{6}$$

The coefficients q_u of the NSPS vector Q^n are equal to the normalized sum of the coefficients r_v for which the implication $\forall i \; u_i \rightarrow v_i$ holds. In general, we are most interested in NSPS transforms of the form $Q(f,f)$, but we will develop the theory for the general case of $Q(f,g)$ since this form is necessary to derive an algorithm for computing $Q(f,f)$.

As an example, consider the boolean function:

$$f = \overline{b}\overline{c}d + b\overline{c}\overline{d} + ab\overline{d} + a\overline{c}d + \overline{a}bcd \tag{7}$$

with the spectral and NSPS coefficient maps shown in Figure 2. The NSPS coefficients are computed from the spectral coefficients which include the same indices as the NSPS coefficient. For example, q_{ad} is the sum of the squares of the Walsh coefficients circled in Figure 2(a), or:

$$\begin{aligned} q_{ad} &= \frac{1}{4}(s^2_{ad} + s^2_{abd} + s^2_{acd} + s^2_{abcd}) \\ &= \frac{1}{4}(2^2 + (-2)^2 + (-6)^2 + 6^2) = 20 \end{aligned}$$

By substituting the Shannon expansion ($m = n - 1$):

$$S^n(f) = \begin{bmatrix} S^m(f_{\overline{x}_m}) + S^m(f_{x_m}) \\ S^m(f_{\overline{x}_m}) - S^m(f_{x_m}) \end{bmatrix} \tag{8}$$

of the Walsh spectrum into (4) and (5), it is relatively easy to derive Shannon expansions for the SPV and the NSPS ($m = n - 1$):

$$R^n(f,g) = \begin{bmatrix} R^m(f_{\overline{x}_m}, g_{\overline{x}_m}) + R^m(f_{\overline{x}_m}, g_{x_m}) \\ \quad + R^m(f_{x_m}, g_{\overline{x}_m}) + R^m(f_{x_m}, g_{x_m}) \\ R^m(f_{\overline{x}_m}, g_{\overline{x}_m}) - R^m(f_{\overline{x}_m}, g_{x_m}) \\ \quad - R^m(f_{x_m}, g_{\overline{x}_m}) + R^m(f_{x_m}, g_{x_m}) \end{bmatrix} \tag{9}$$

$$Q^n(f,g) = \begin{bmatrix} Q^m(f_{\overline{x}_m}, g_{\overline{x}_m}) + Q^m(f_{x_m}, g_{x_m}) \\ Q^m(f_{\overline{x}_m}, g_{\overline{x}_m}) - Q^m(f_{\overline{x}_m}, g_{x_m}) \\ \quad - Q^m(f_{x_m}, g_{\overline{x}_m}) + Q^m(f_{x_m}, g_{x_m}) \end{bmatrix} \tag{10}$$

If we define $\mathtt{sup}(f)$ as the set of variables in the support of f, then it is easy to show that if $x_i \notin \mathtt{sup}(f) \cap \mathtt{sup}(g)$, and $u_i = 1$, then $\xi_u \cdot Q^n(f,g) = 0$ by noting that $x_i \notin \mathtt{sup}(f)$

implies that $f_{x_i} = f_{\overline{x}_i}$. Some additional useful properties of $Q^n(f,g)$ are:

$$\begin{aligned}
&Q^n(f,g) = Q^n(g,f) = Q^n(\overline{f},\overline{g}) \\
&Q^n(f,g) = -Q^n(f,\overline{g}) = -Q^n(\overline{f},g) \\
&\xi_0 \cdot Q^n(f,g) = Y^n(f) \cdot Y^n(g) \\
&\xi_0 \cdot Q^n(f,f) = 2^n \\
&Q^0(1,1) = 1
\end{aligned} \tag{11}$$

4 Function Complexity

In logic synthesis it is often important to estimate the complexity of a boolean function. One estimator frequently used with spectral translation methods is the complexity factor $C^n(f)$ equal to the number of vector pairs (X_1, X_2) for which $f(X_1) = f(X_2)$ and for which the Hamming distance $d_H(X_1, X_2) = 1$. This estimator is a scalar value in the range 0 to $n2^n$. Large values of $C^n(f)$ imply a simpler AND/OR decomposition with $C^n(1) = n2^n$ and $C^n(x_0 \oplus x_1 \oplus \cdots \oplus x_{n-1}) = 0$. The complexity factor is invariant with respect to NPN-equivalent functions[6]. NPN-equivalent functions are functions which can be formed by Negation of inputs, Permutation of inputs or Negation of the output. It has been shown that complexity can be written as a weighted sum of squares of the Walsh spectral coefficients [6]:

$$C^n(f) = n2^n - \frac{1}{2^n} \sum_{u=0}^{2^n-1} ||u|| s_u^2 \tag{12}$$

This equation gives us a theoretical justification as to why functions that have high-magnitude spectral coefficients at low orders have better AND/OR representations. The complexity $C^n(f)$ of an n-input function f can also be written as the dot product:

$$C^n(f) = n2^n - \frac{1}{2} E^n \cdot Q^n(f,f) \tag{13}$$

where E^n is a 2^n element vector $E^n = \sum_{i=0}^{n-1} \xi_{2^i}$ in which the first-order elements are 1 and all other elements are 0. The term $E^n \cdot Q^n(f,f)$ is the sum of all first-order NSPS coefficients. We can show that (13) is equivalent to (12) by noting that $E^n \cdot Q^n(f,f)$ can be expanded to $\frac{1}{2^{n-1}} \sum_{i=0}^{n-1} \sum_{u=0}^{2^n-1} s_u^2 u_i$ which can also be written as $\frac{1}{2^{n-1}} \sum_{u=0}^{2^n-1} s_u^2 \sum_{i=0}^{n-1} u_i$. From the definition of $||u||$, we can write this as $\frac{1}{2^{n-1}} \sum_{u=0}^{2^n-1} ||u|| s_u^2$ which when substituted back into (13) gives us (12).

5 Spectral Translation

In this section we will investigate the effect of a translation $f_{x_i = x_i \oplus x_j}$ on the NSPS and on the complexity of a boolean function. For brevity, vector function arguments and dimensions will be omitted where obvious, and a prime will be used to indicate the vectors for translated functions (e.g., S' for $S^n(f_{x_i = x_i \oplus x_j})$). We begin by defining the $2^n \times 2^n$ translation matrix $P_{i,j}^n$ such that:

$$Y^n(f_{x_i = x_i \oplus x_j}) = P_{i,j}^n Y^n(f) \tag{14}$$

This matrix transforms the truth table of f to the truth table of $f_{x_i = x_i \oplus x_j}$. The translated truth table can be obtained by exchanging truth table entries where x_i is 1 with those where x_i is 0 whenever x_j is 1. The matrix which describes this transformation is:

$$P_{i,j}^n = \sum_{u=0}^{2^n-1} \xi_u \xi_u^T \overline{u_j} + \xi_u \xi_{u+2^i}^T u_j \overline{u_i} + \xi_u \xi_{u-2^i}^T u_j u_i \tag{15}$$

For example, the translation matrix for $f_{x_{n-2} = x_{n-2} \oplus x_{n-1}}$ is:

$$P_{n-2,n-1}^n = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & 0 & I \\ 0 & 0 & I & 0 \end{bmatrix} \tag{16}$$

We now propose the following theorem for the change in Q as a result of a translation.

Theorem 1 *For any n-input boolean functions f and g:*

$$Q^n(f_{x_i = x_i \oplus x_j}, g_{x_i = x_i \oplus x_j}) = T_{i,j}^n Q^n(f,g) \tag{17}$$

where $i \neq j$ and where $T_{i,j}^n$ is:

$$T_{i,j}^n = K^n W^n P_{i,j}^n (W^n)^{-1} (K^n)^{-1} \tag{18}$$

Proof: The transformation (18) is simply a mapping of $P_{i,j}^n$ from the boolean domain to spectral domain and then to NSPS domain. We can show this by expanding (17) to obtain $KR' = (KWP_{i,j}W^{-1}K^{-1})(KR)$ which we can reduce to $R' = WP_{i,j}\, W^{-1}R$. Since there is a direct mapping between elements of $R^n(f,g)$ and elements of $S^n(f)$ and $S^n(g)$, it also holds that $S' = WP_{i,j}W^{-1}S$. Expanding this we get $WY' = (WP_{i,j}W^{-1})(W\,Y)$ which can be simplified to $Y' = P_{i,j}Y$ which holds by the definition of $P_{i,j}^n$. □

For the translation in the previous example, the matrix $T_{i,j}^n$ has the form:

$$T_{n-2,n-1}^n = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & I & I & -I \\ 0 & 2I & 0 & -I \end{bmatrix} \tag{19}$$

thus it is easy to see that the only first and second-order coefficients to change for $k \neq i \neq j$ are q_{x_j}, $q_{x_i x_j}$ and $q_{x_j x_k}$, and that:

$$\begin{aligned}
q'_{x_j} &= q_{x_i} + q_{x_j} - q_{x_i x_j} \\
q'_{x_i x_j} &= 2q_{x_i} - q_{x_i x_j}
\end{aligned} \tag{20}$$

The change in complexity resulting from the same translation is described by the following theorem.

Theorem 2 *The increase in the complexity of the n-input function f resulting from replacing input x_i with $x_i \oplus x_j$ where $i \neq j$ is:*

$$\Delta C_{i,j}^n(f) = \frac{1}{2}(\xi_{2^i+2^j} - \xi_{2^i}) \cdot Q^n(f,f) \tag{21}$$

Proof: From the definition of complexity (13), and the Theorem 1, we can rewrite the change in complexity as:

$$\begin{aligned}
\Delta C_{i,j}^n(f) &= C^n(f_{x_i = x_i \oplus x_j}) - C^n(f) \\
&= \frac{1}{2}((E^n)^T (I - T_{i,j}^n)) \cdot Q^n(f,f)
\end{aligned} \tag{22}$$

Since complexity is invariant for any ordering of the input variables, we only need to show (22) for one pair (i,j) where $i \neq j$. For convenience, we choose $i = n-2$ and $j = n-1$. Since all rows u of $(I - T_{n-2,n-1}^n)$ for $u < 2^{n-1}$ are zero, and the only non-zero element of E^n for $u \geq 2^{n-1}$ is at $u = 2^{n-1}$, the expression $(E^n)^T (I - T_{n-2,n-1}^n)$ reduces to row $u = 2^{n-1}$ of $(I - T_{n-2,n-1}^n)$, which is $\xi_{2^{n-2}+2^{n-1}} - \xi_{2^{n-2}}$, or in the general case $\xi_{2^i+2^j} - \xi_{2^i}$. Substituting this back into (22) we get (21). □

```
Translate(f, n)
{
  f'=f;
  σ=I;
  for (0 ≤ i ≤ n − 1, i + 1 ≤ j ≤ n − 1)
    Q[i][j]=Q[j][i]=NSPS(f, f, 2^i + 2^j, n);
  for (0 ≤ i ≤ n − 1)
    Q[i][i]=NSPS(f, f, 2^i, n);
  while (∃i, j  Q[i][j] − Q[i][i] > 0) {
    pick i and j (i ≠ j) maximizing Q[i][j] − Q[i][i];
    f'=f'_{x_i = x_i ⊕ x_j}
    xor row j of σ into row i;
    Q[j][j]=Q[j][j] + Q[i][i] − Q[i][j];
    Q[j][i]=Q[i][j]=2 * Q[i][i] − Q[i][j];
    for (0 ≤ k ≤ n − 1,  k ≠ i, j)
      Q[j][k]=Q[k][j]=NSPS(f', f', 2^j + 2^k, n);
  }
  return (σ, f');
}
```

Figure 3: Spectral Translation Algorithm

6 Implementation and Example

The BDD package used in implementing the techniques described here is based on [2] and uses compliment edges. To simplify indexing in the algorithms presented here, nodes are indexed with the highest index at the top of the graph, decreasing towards the terminal node. The index of the lowest variable is zero. We use the following notation in the algorithms:

`isterm`(f) Returns true if f is a terminal node.

$T(f)$ Returns the index of the top variable, or -1 if f is a terminal node.

$H(f)$ Returns the 1-edge of the top node, or f if it is terminal node.

$L(f)$ Returns the 0-edge of the top node, or f if it is terminal node.

$H_t(f)$ Returns $H(f)$ if $T(f) = t$, otherwise returns f.

$L_t(f)$ Returns $L(f)$ if $T(f) = t$, otherwise returns f.

`max`(a, b) Returns the maximum of integers a and b.

$B_i(u)$ Returns 1 if bit i of u is set, otherwise returns 0.

`mssb`(u) Return the index of the most significant set bit. That is, the largest i for which $B_i(u)$ is 1.

`cmssb`(u) . Clear most significant bit for which $B_i(u)$ is 1. $u - 2^{\mathtt{mssb}(u)}$ if $u \neq 0$, or otherwise undefined.

The main synthesis algorithm (Figure 3) is `Translate` which takes a single function f represented as a BDD, and the number of inputs n, and returns the pair (σ, f') such that $f(X) = f'(\sigma X)$ where σ is an $n \times n$ matrix representing the linear block, and f' is the non-linear block represented as a BDD. In our implementation, we choose to translate each output separately, and merge the results after translation. For a multi-output function, this could result in an increase in the number of inputs to the merged non-linear block, but in practice few additional inputs are needed. An alternative approach would be to use a combined spectrum for all of the outputs by adding the individual Y vectors for each output function and applying the algorithms presented in this paper by using an MTBDD (Multi-Terminal BDD) based representation. This approach, however, can lead to inefficient representations, especially when the outputs have different support. Another problem with the combined spectrum method is that different output functions may be optimal under different bases and it may not be possible to optimize all functions with the same translations.

1) $f' = \overline{b}\overline{c}d + b\overline{c}\overline{d} + ab\overline{d} + a\overline{c}d + \overline{a}bcd$ $\quad C^4(f') = 28$

Q	1	a	b	c	d
a	12	-	12	$\underline{20}$	$\underline{\underline{20}}$
b	20	12	-	20	$\underline{28}$
c	20	20	20	-	$\underline{28}$
d	20	20	$\underline{28}$	$\underline{28}$	-

σ	a	b	c	d
a	1	0	0	0
b	0	1	0	0
c	0	0	1	0
d	0	0	0	1

2) $f' = \overline{b}\overline{c}d + b\overline{c}\overline{d} + \overline{a}\overline{c}d + abc$ $\quad C^4(f') = 32$

Q	1	a	b	c	d
a	12	-	12	$\underline{\underline{20}}$	4
b	20	12	-	20	20
c	20	20	20	-	12
d	12	4	$\underline{20}$	12	-

σ	a	b	c	d
a	1	0	0	1
b	0	1	0	0
c	0	0	1	0
d	0	0	0	1

3) $f' = \overline{b}\overline{c}d + b\overline{c}\overline{d} + \overline{a}b$ $\quad C^4(f') = 36$

Q	1	a	b	c	d
a	12	-	12	4	4
b	20	12	-	12	20
c	12	4	12	-	12
d	12	4	$\underline{\underline{20}}$	12	-

σ	a	b	c	d
a	1	0	1	1
b	0	1	0	0
c	0	0	1	0
d	0	0	0	1

4) $f' = \overline{a}b + \overline{c}d$ $\quad C^4(f') = 40$

Q	1	a	b	c	d
a	12	-	12	4	4
b	12	12	-	4	4
c	12	4	4	-	12
d	12	4	4	12	-

σ	a	b	c	d
a	1	0	1	1
b	0	1	0	0
c	0	0	1	0
d	0	1	0	1

Figure 4: Spectral Translation Example

The algorithm proceeds by first initializing f' to the function to be translated, and σ to the identity matrix. The $n \times n$ array `Q[][]` is then initialized with the first and second order coefficients of $Q^n(f, f)$. Since this array is symmetric, only the n first order coefficients, and the $(n^2 - n)/2$ second order coefficients need to be computed. The algorithm then loops until there is no improvement in the complexity. On each iteration (21) is used to select the translation $x_i = x_i \oplus x_j$ having the largest improvement. A composition operation is then done to update f', and the translation is recorded in σ by XORing row j into row i. The coefficients $q_{x_i x_j}$ and q_{x_j} are updated using (20), and finally, the $n - 2$ spectral product coefficients $q_{x_j x_k}$, $(i \neq j \neq k)$ are recomputed.

For example, consider the execution of `Translate` (Figure 4) on the boolean function in (7). For each iteration, the values of f', σ, `Q` and $C^4(f')$ are shown at point where the `while` loop condition is tested. Values in the "1" column of `Q` are the first order spectral coefficients $\xi_{2^i} \cdot Q^4(f, f)$ where i is the index for each of the input variables a, b, c and d. The remaining columns of `Q` are the second order spectral coefficients $\xi_{2^i+2^j} \cdot Q^4(f, f)$ where i and j are the indices for two different input variables. Underlined values in `Q` represent candidate translations with a positive ΔC. Double underlined values represent the value actually selected for translation. For example, in the first iteration of the loop the translation $a = a \oplus d$ with a $\Delta C = \frac{1}{2}(20 - 12) = 4$ was selected. In this case, all the underlined candidate translations had the same ΔC so one was chosen at random. Subsequent iterations were performed in a like manner obtaining the

```
NSPS(f, g, u, n)
{
  if (isterm(f)&&isterm(g)) return 2^n * f * g;
  if (u==0) return Y^n(f) · Y^n(g);
  l=mssb(u);
  t=max(T(f), T(g));
  u'=cmssb(u);
  if (l > t || (t==l&&(l!=T(f) || l!=T(g))))
    return 0;
  if (([f, g, u] → q) in compute table) return q * 2^(n-t+1);
  q=NSPS(L_t(f), L_t(g), u', t) + NSPS(H_t(f), H_t(g), u', t);
  if (t == l)
    q=q − NSPS(L_t(f), H_t(g), u', t) − NSPS(H_t(f), L_t(g), u', t);
  store ([f, g, u] → q) in compute table;
  return q * 2^(n-t+1);
}
```

Figure 5: The NSPS Algorithm

translations $a = a \oplus c$ in Step 2, and $d = d \oplus b$ in Step 3. In Step 4, there were no more translations for which ΔC was positive, so the algorithm was terminated. By applying the algorithm, the original function with 16 literals in the SOP form was reduced to a function with only 4 literals.

Figure 3 shows the basic algorithm based on (10) for computing the NSPS coefficient $\xi_u \cdot Q^n(f, g)$. When $u = 0$, the NSPS coefficient reduces to $Y^n(f) \cdot Y^n(g)$, which can be computed using the decomposition($m = n - 1$):

$$Y^n(f) \cdot Y^n(g) = Y^m(f_x) \cdot Y^m(g_x) + Y^m(f_{\overline{x}}) \cdot Y^m(g_{\overline{x}})$$

The indices l and t are respectively the highest index included in u, and the index of the variable with the highest index in the union of the supports of f and g. If $l > t$, or $l = t$ and the top variable of neither f nor g is l, then u contains variables which are not in the support of f or g, and thus the result is 0. Otherwise, the spectral coefficient is computed from the recursive definition of the NSPS and multiplied by 2^{n-t+1} where is $n-t+1$ represents the number of skipped variables or "cross points"[9]. The hit rate of the computed table can also be improved by utilizing the symmetry properties shown in (11).

7 Experimental Results

The algorithms described here have been applied to a number of benchmark circuits shown in Table 1. A number in parenthesis after the number of input terminals indicates the number of inputs in the support of the output with the largest support. The number of BDD nodes, and the number of terms and literals in the SOP form is shown before and after translation. The term and literal count is obtained by converting the BDDs to ZBDDs (zero-suppressed BDD) [7] using an algorithm that generates irredundant SOP forms from the BDD, and then using ZBDD algorithms for counting literals and terms. For each output, both before and after translation, the phase that resulted in the smallest SOP form (by literal count) was used. The last four columns represent the cost of the translation. "trans" is the number of translation operations that were performed; "xor" is the number of XOR gates in the optimized tree; "+in" is the number of additional input pins to the non-linear block; and "cpu" is the run-time in seconds on a SPARCstation-20. The linear block was optimized using a simple rectangle covering algorithm to identify common terms, but we expect that a more sophisticated algorithm employing symmetry properties of the XOR operation could achieve even smaller linear blocks.

Using our methods, we have been able to apply spectral translation to circuits with many more inputs and much larger cube sets than previously possible. While an increase in the complexity factor is not guaranteed to result in a reduction in SOP literals, a reduction was obtained for nearly all of the examples we tested. The circuit "des" is the Data Encryption Standard chip for which we reduced the number of literals by over a factor of 100. For the circuit "ham16", a 16-bit hamming distance circuit, we achieved a reduction of nearly four orders of magnitude. Also of interest is the parity function "parity" which can be implemented as a single tree of XOR gates.

The results of synthesizing and mapping several benchmark circuits are shown in Table 2. We synthesized the unmodified circuit and the non-linear block of the translated circuit under SIS 1.3 using `script.rugged`. Mapping was done using the library `lib2.mcnc2lib` with the command `map -m 0`. For the translated circuit we proceeded by using the rugged script on the translated portion, attaching the linear block, and then mapping the combined circuit. The table shows that the area[2] and delay before and after using Spectre.

Functions that exhibit a large reduction in the number of literals of the SOP form with a small number of XOR gates often yield a more efficient circuit when spectral translation is applied. For example, by applying Spectre to the benchmark "sec", a single-error-correct/double-error-detect encode/decode circuit, we cut both delay and area by nearly half. But when we synthesized and mapped the example function, we note that while the reduced form looks much simpler than the original function, when we actually do the mapping, we find that we get an area of 25 and a delay of 5.06 for the original circuit, and an area of 26 and a delay of 5.07 for the Spectre processed circuit. One reason for this is that while the three XOR gates added in the translation process reduced the number of literals by a factor of four, the actual number of literals reduced was not very large, thus the cost of the XOR gate is not justified by the relatively small absolute reduction in literals. Another factor is that for smaller functions, multi-level synthesis systems can search a much larger portion of the search space, further reducing the benefit of using spectral translation techniques.

We were also unable to obtain a significant improvement in circuit size or delay for large functions such as the "des" description. While there was a drop by two orders of magnitude in the number of literals of this function as a result of spectral translation, we obtained an area of 6213 for the original circuit, and an area of 7824 (5684 for the non-linear block and 2140 for the linear block) for the spectrally translated circuit. We believe the main reason for this is that since Spectre generates f' directly from a BDD or ZBDD (by replacing each node with the boolean function represented at that node), the function is essentially flattened before synthesis, whereas SIS (with `script.rugged`) does not collapse the network before synthesis. This puts Spectre generated output at a disadvantage when the original input function has good global structure. Clearly, it is necessary to apply methods for improving the form of the non-linear block to obtain better synthesis results. Recent work for producing good multi-level forms from BDDs or ZBDDs[8] looks promising to this end.

[2] Area measurements are divided by 464, the greatest common factor of all the library components.

Circuit			Before Trans.			After Trans.			Cost			
name	in	out	bdd	term	lit	bdd	term	lit	trans	xor	+in	cpu
example	4	1	8	4	10	9	3	8	1	3	0	0.1
aluA	19	8	300	2552	26025	634	1568	13410	64	64	49	19.23
apex6	135(24)	99	1256	656	3222	1172	588	2292	39	28	28	27.46
des	256(19)	245	7388	203631	2584463	4887	4301	22633	1087	428	181	293.6
f51m	8	8	39	76	323	54	40	123	24	19	14	1.26
frg2	143(25)	139	3757	3656	18622	3685	3449	17434	135	17	13	95.91
gcd	12	6	813	767	5201	741	335	3059	81	60	37	40.85
ham8	16	4	116	46608	711200	36	151	941	39	15	8	2.31
ham16	32	5	456	3.0e9	5.8e9	144	50323	657457	95	31	16	17.95
hidden	8	1	47	67	423	58	41	227	8	8	0	0.99
k2	45(39)	45	1656	923	6851	1916	924	6362	208	86	75	136.03
lal	26(13)	19	101	99	239	91	83	196	12	12	8	2.78
ldd	9	19	80	59	206	81	54	177	10	10	5	2.32
mul	8	8	171	148	774	193	135	630	19	14	14	3.09
myadd	33	17	457	655287	1.1e7	486	393178	6.2e6	32	32	31	12.22
parity	16	1	17	32768	524288	2	1	1	15	15	0	0.37
pcle	19(12)	9	81	45	185	88	38	155	7	7	7	1.37
rd53	5	3	17	31	140	14	16	61	4	4	1	0.32
rd73	7	3	31	147	876	26	84	429	6	6	1	0.41
rd84	8	4	42	278	1964	36	151	941	7	7	1	0.55
sec	8	8	109	125	697	111	80	381	14	13	10	1.94
term1	34(20)	10	520	257	1501	552	192	1143	20	17	10	20.13
ttt2	24(14)	21	182	136	489	176	123	390	20	18	16	4.08
vda	17	39	595	558	3104	708	471	2343	219	57	54	31.54
z4ml	7	4	23	59	252	21	32	117	6	6	5	0.39
c1908	33(32)	25	12712	3.6e7	9.4e8	19682	1641147	42981741	399	228	104	2762
c880	60(44)	25	3930	73312	1101942	19145	71110	1055984	93	74	58	2772

Table 1: Experimental Results on Benchmark Circuits

	Before Trans.		After Trans.	
name	area	delay	area	delay
example	25	5.06	26	5.07
sec	674	23.02	357	13.95
rd53	67	10.84	63	7.68
rd74	125	18.73	167	12.02
des	6213	126.36	7824	117.53

Table 2: Experimental Results of Mapped Circuits

8 Conclusion

In this paper we have presented a new algorithm (Spectre) for spectral translation using boolean decision diagrams. The algorithm is based on using sums of squares of spectral coefficients and uses an efficient BDD based algorithm to compute the coefficients. While the effectiveness of spectral translation is highly dependent on the function being synthesized, we have shown that only the first and second order sum of squares coefficients are required to achieve a significant reduction in the number of literals and terms in the SOP form for many functions.

References

[1] Bryant, R,E., "Graph-Based Algorithms for Boolean Function Manipulation", *IEEE Transactions on Computing* , C-38, 8, pp. 677-91, (August 1986)

[2] Brace, K.S., Rudell, R.L and Bryant, R.E., "Efficient Implimentation of a BDD Package", From *27th ACM/IEEE Design Automation Conference* , pp. 40-5 (1990)

[3] Brayton, R.K., et. al., "MIS: A Multiple-Level Logic Optimization System", From *IEEE Transactions on Computer-Aided Design*, vol. 6, No. 6, pp. 1062-1081 (1987)

[4] Clark, E.M., et. al., "Spectral Transforms for Large Boolean Functions with Applications to Technology Mapping", From *30th ACM/IEEE Design Automation Conference*, pg. 54-60 (1993)

[5] Falkowski, B.J. and Chang C.H., "Efficient Algorithms for the Calculation of Arithmetic Spectrum from OBDD and Synthesis of OBDD from Arithmetic Spectrum for Incompletely Specified Boolean Functions", From *IEEE International Symposium on Circuits and Systems* vol. 1, pg. 197-200 (1994)

[6] Hurst, S.L., Miller, D.M. and Muzio J.C., "Spectral Techniques in Digital Logic", Academic Press (1985)

[7] Minato, Shin-ichi, "Zero-Suppressed BDDs for Set Manipulation in Combinational Problems", From *30th ACM/IEEE Design Automation Conference*, pg. 272-277 (1993)

[8] Minato, Shin-ichi, "Fast Weak-Division Method for Implicit Cube Representation", From *Synthesis And System Integration of Mixed Technologies (SASIMI)*, pg. 423-432 (1993)

[9] Stanković, R.S., "Some Remarks about Spectral Transform Interpretation of MTBDDs and EVBDDs", From *The Asia and South Pacific Design Automation Conference*, pg. 385-390 (1995)

[10] Varma, D., Trachtenberg, E.A., "Design Automation Tools for Efficient Implementation of Logic Functions by Decomposition", From *IEEE Transactions on Computer-Aided Design*, vol. 8., no 8., pg. 901-916 (1989)