It is possible to further increase the speed performance of the modified LEAP circuit. As mentioned before, the selection of input variable order for the construction of a BDD is usually based on the criterion of minimising the number of nodes. However, in practice, designers may define their own variable order according to a realistic implementation situation. For example, if not all the input variables of a Boolean function arrive at the same time, one might give the slowest input variable the highest priority during Shannon expansion so that the path from this primary input to the output passes fewest multiplexers. Another possible improvement is to construct a BDD-like tree with each node implemented by a higher-order multiplexer. For example, in Fig. 3a, a radix-2 BDD tree of height 20 has a critical path going through 20 two-to-one multiplexers plus eight-inverters. Conversely, the radix-4 BDD-like tree of height 10 has the critical path going through only 10 four-to-one multiplexers plus five-inverters, a significant improvement in speed performance. Fig. 3b shows the four-to-one multiplexer used in the radix-4 implementation.

In fact, the BDD-based synthesis process can combine multiplexers of different orders in order to achieve optimal speed performance. For example, the nodes in upper levels can be realised using high-order multiplexers while low-order multiplexers implement the nodes in lower levels. Hence, the larger horizontal delay through the controlling input to high-order multiplexers in the upper levels is hidden under the vertical delay through the pass-transistor chain and inverters in lower levels.

Conclusion: We present a new top-down cell-based design method based on pass-transistor logic with only two types of cell, multiplexers and inverters, in the cell library. The corresponding logic circuit synthesis tool is also developed, which takes advantage of the characteristics of pass-transistor logic to minimise the delay and area cost. Further performance improvement can be achieved using mixed-radix design and the option of user-defined variable order.

Algorithm for cosine transform of Toeplitz matrices

F. Argüello, M.A. Trenas, J. López and E.L. Zapata

An algorithm for calculating the 2D cosine transform of a Toeplitz matrix is presented. The algorithm is based on the application of 1D cosine transforms. More specifically, four 1D cosine transforms of size $N$ are needed to obtain the transform of a Toeplitz matrix of size $N \times N$. This is an improvement over previously published algorithms (for example, [1] requires five Fourier transforms of size $2N$). The algorithm is also simple and regular.

Introduction: Toeplitz matrices arise in many applications of digital signal and image processing. In [1], Ohsmann presents an algorithm for computing the cosine transform (DCT) of a Toeplitz matrix. The derivation of the algorithm is based on the mapping of a 1D DCT of size $N$ on a 1D fast Fourier transform (FFT) of size $2N$. The algorithm requires the evaluation of five FFTs of this type. The author also presents some applications of this algorithm: eigenvalue computation, matrix approximation and subspace selection, regularisation, and estimation of condition numbers. These applications are mainly based on the concentration property of the transforms involved. In [2], the results are generalised in order to consider arbitrary fast transforms.

In this Letter we present a similar algorithm which requires fewer arithmetic operations. The algorithm is based on the use of 1D DCTs in place of 1D FFTs.

Derivation of the algorithm: A Toeplitz matrix of size $N \times N$ has the form

$$
a = \begin{pmatrix}
a_0 & a_1 & a_2 & \ldots & a_{N-1} \\
a_{-1} & a_0 & a_1 & \ldots & a_{N-2} \\
a_{-2} & a_{-1} & a_0 & \ldots & a_{N-3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
a_{-N+2} & a_{-N+1} & a_{-N} & \ldots & a_0
\end{pmatrix}
$$

This matrix can be represented as $a = (a_i)$, $a_i = a_{i-N}$, in which the input data are $a_i, i = -N+1, \ldots, N-1$ (2N - 1 input data). In the algorithm which we derive, the variables $a_{N}$ and $a_{0}$ also appear as input data, but the value of these variables does not interfere with the final result. Setting $a_{N} = a_{0} = 0$ is numerically preferable.

The DCT $A = (A_{m,n})$ of a 2D matrix $a = (a_{i})$ (not necessarily a Toeplitz matrix) of size $N \times N$ is given by the following expression:

$$
A_{m,n} = r_{mn} \left( \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} c_{m,l} a(k, l) C_{N,k,l} \right) \tau_{n}
$$

where

$$
c_{m,l} = \cos \left( \frac{(2l+1)m \pi}{2N} \right)
$$

References

\[ \tau_m = \begin{cases} \sqrt{\frac{3}{n}} & m = 0 \\ \sqrt{\frac{2}{n}} & m = 1, 2, \ldots, N - 1 \end{cases} \] (4)

The simplest way of calculating a bidimensional transform is by breaking it down into multiple unidimensional transforms on rows and columns. Calculated in this manner, it requires 2N 1D DCTs (one for each row and each column). The algorithmic complexity of each of these 1D DCTs is O(N log N). Furthermore, in the literature, other more efficient algorithms have been proposed for the calculation of a bidimensional transform. For example, for the case of the cosine transform, \([3]\) reduces the number of 1D DCTs required to N.

Nevertheless, the calculation can be simplified in a much more radical fashion by using the special form of the Toeplitz matrix. We will demonstrate that, by this method, the number of 1D DCTs needed can be reduced to four. The method is based on eliminating the redundant operations that take place in the realization of the unidimensional transforms carried out on the columns (first step) and on the rows (second step) of the Toeplitz matrix.

The first step, realized directly, involves computing the 1D DCT for each column of the Toeplitz matrix (which we denote \(x_{m,n} \), \(0 \leq k < N\)).

\[ x_{m,k} = \sum_{l=0}^{N-1} a_{k-l} \cos(l + 0.5) \theta_m \] (5)

with \( \theta_m = \pi mN \).

\[ x_{m,k+1} = \sum_{l=0}^{N-1} a_{k-l+1} \cos(l + 0.5) \theta_m \] (6)

The special form of the Toeplitz matrix allows us to relate the 1D DCTs of the different columns. In our case, we do not apply the breakdown \(\exp(a + b) = \exp(a) \exp(b)\), as is carried out in [1]. Nevertheless, we can apply the formula

\[ \cos(a + b) = 2 \cos(a) \cos(b) - \cos(a - b) \] (7)

By applying this formula the relation between the DTCs of the successive columns of the Toeplitz matrix is

\[ x_{m,k+1} + x_{m,k-1} - x_{m,k} 2 \cos \theta_m = \{ (a_{k+1} - a_k) + (-1)^m (a_{k-N} - a_{k-N+1}) \} \cos(\theta_m/2) \) (8)

The second step computes the 1D DCTs of the rows. Carried out directly, this involves computing N 2D DCTs \(A_{n,m}, 0 \leq n, m < N\), using the formula

\[ A_{m,n} = \sum_{k=0}^{N-1} x_{m,k} \cos(k + 0.5) \theta_n \] (9)

Nevertheless, due to the special form of the Toeplitz matrix, it is not necessary to compute any of these N transforms. The 2D DCT of the Toeplitz matrix can be obtained by way of the formula

\[ A_{m,n,2} \cos \theta_n - \cos \theta_m = \{ u_1(n) + (-1)^{n+m+1} u_2(n) \} \cos(\theta_m/2) + \{ u_2(n) + (-1)^{n+m+1} u_1(n) \} \cos(\theta_n/2) \] (10)

where

\[ u_1(j) = \sum_{k=0}^{N-1} (a_{k-j} + a_{k-j-1}) \cos(k + 0.5) \theta_j \] (11)

\[ u_2(j) = \sum_{k=0}^{N-1} (a_{k+j} - a_k) \cos(k + 0.5) \theta_j \] (12)

This equation enables us to obtain the 2D DCT, \(A_{m,n}\), for the co-ordinates \(n \neq m\). \(u_1(0)\) and \(u_2(0)\) (\(i = 0, 1, \ldots, N - 1\)) are the only 1D DCTs that need to be computed in this case.

For the points of the co-ordinates \(n = m\) we cannot apply eqn. 10 as this results in an indetermination. By using the Hôpital’s rule we derive the expression with respect to \(\theta_m\), maintaining \(\theta_n\) constant, and later we substitute \(n = m\). Then we obtain

\[ A_{m,m,2} \sin \theta_m = 0.5 u_1(m) - 0.5 \cos 2 \theta_m \sin \theta_m/2 + u_1(N - m) \cos(\theta_m/2) \]

(13)

where

\[ u_3(m) = \sum_{k=0}^{N-1} (a_{k+1} + a_k) \cos(k + 0.5) \theta_m \] (14)

\[ u_4(m) = \sum_{k=0}^{N-1} (-1)^k (k + 0.5)(a_{k+1} - a_k) + (N - k - 0.5)(a_{m-k} + a_{m-k-1}) \cos(k + 0.5) \theta_m \] (15)

This formula enables us to obtain \(A_{m,n}, m \neq 0\) requiring only the computation of the two additional 2D DCTs, \(u_3(m)\) and \(u_4(m)\).

The case \(n = m = 0\) once again constitutes an indetermination. Nevertheless, it can be immediately calculated by using the expression

\[ A_{0,0} = N \theta_0 + \sum_{k=1}^{N-1} (N - k) (a_k + a_{-k}) \] (16)

which has a computational complexity of \(O(N^2)\).

Algorithm: In summary, the proposed algorithm for computing the 2D DCT of a Toeplitz matrix involves the following steps:

(i) Computation of four 1D DCTs using eqns. 11, 12, 14 and 15. These can be computed, for example, by using the algorithms [4–6].

(ii) Obtaining \(A_{m,n}\) for the points of co-ordinates \(n \neq m\) by using eqn. 10; for the points \(n = m \neq 0\) use eqn. 13; and for \(n = m = 0\) use eqn. 16.

In total we need \(5N + 3(N^2 - 1)\) additions, \(2N^2 - 1\) products by trigonometric coefficients and \(3N^2\) products from scale factors, as well as four 1D DCTs of size \(N\). This is a notable reduction with respect to [1], which amongst other operations needs to compute five 1D FFTs of size \(2N\).

Conclusions: We have presented an algorithm to compute the 2D cosine transform of a Toeplitz matrix. The algorithm exploits the special form of this type of matrix and basically requires the evaluation of four 1D cosine transforms. The algorithm obtained requires less operations and is simpler and more regular than previously proposed algorithms.

Acknowledgments: This work was supported in part by the Ministry of Education and Science (CICYT) of Spain under contract TIC96-1125-C03, Xunta de Galicia XUGA20605B96 and EC project BRPR-CT96-01070.