Power Efficient Hierarchical Scheduling for DSP Transformations

P. K. MERAKOS*, K. MASSELOS and C. E. GOUTIS

VLSI Design Lab., Department of Electrical and Computer Engineering, University of Patras, Rio 26500, Greece

(Received 1 April 2000; Revised 23 May 2000)

In this paper, the problem of scheduling the computation of partial products in transformational Digital Signal Processing (DSP) algorithms, aiming at the minimization of the switching activity in data and address buses, is addressed. The problem is stated as a hierarchical scheduling problem. Two different optimization algorithms, which are based on the Travelling Salesman Problem (TSP), are defined. The proposed optimization algorithms are independent on the target architecture and can be adapted to take into account it. Experimental results obtained from the application of the proposed algorithms in various widely used DSP transformations, like Discrete Cosine Transform (DCT) and Discrete Fourier Transform (DFT), show that significant switching activity savings in data and address buses can be achieved, resulting in corresponding power savings. In addition, the differences between the two proposed methods are underlined, providing envisage for their suitable selection for implementation, in particular transformational algorithms and architectures.

Keywords: Low power design and consumption; Hierarchical scheduling; Sum-of-product computation; Digital signal processing; High level synthesis

INTRODUCTION

During the last years the constrained of power consumption in the design of digital VLSI circuits has earned increased importance, almost in every domain of electronic products.

Portability issues demand low power consumption, because of limited battery energy capacities. Increased battery life means greater weight, causing a market disadvantage for portable products. The power consumption is also important for non-portable computing products, due to the increased cost of cooling and packaging [1].

Increased performance is substantial for every electronic computing product, due to the more demanding applications and greater competition. This is especially true for multimedia and Digital Signal Processing (DSP) systems. Such systems span every market segment. A mean for improving the performance of these systems is to increase their operating frequencies. Taking into account that power consumption is directly proportional to the operating frequency [2], it is very important to find ways that reduce power consumption without sacrificing performance.

Commonly used DSP and multimedia algorithms/processing require the application of transformations like

Discrete Cosine Transform (DCT) and Discrete Fourier Transform (DFT). A matrix–vector multiplication type of computation characterizes these transformations. The constant coefficients, concerning the algorithms, are stored in the matrix, while the data that are processed, are inserted dynamically in the vector, in sequential steps of computation.

The data-path functional unit that implements in hardware the above type of computation is the Multiply Accumulate (MAC) unit. Each element of the coefficient matrix is multiplied with the corresponding data element and the result is accumulated until the inner product is computed. Each intermediate product is called partial product.

In this paper a power efficient scheduling of the multiply-accumulate operations, which are needed in transformational algorithms, is presented. This scheduling is, in effect, a re-ordering of the conventional sequence of computations that are needed for matrix–vector multiplication. Two different re-orderings are obtained, by using two combinatorial optimization algorithms, which are based on Travelling Salesman Problem (TSP). The cost function that drives the optimization procedure is the Hamming Distance (HD) of both, the coefficients and

*Corresponding author. Tel.: +30-61-997-324. Fax: +30-61-994-798. E-mail: merakos@vlsi.ee.upatras.gr

data, which are inserted on the inputs of MACs. Minimizing the switching on the inputs of MACs is equivalent to minimizing the switching on the buses, connecting the MACs with the coefficients and data memories. This substantially reduces the active capacitance, (capacitance that is switched), since buses are very significant capacitive loads [3]. Consequently, recalling that power and active capacitance are directly proportional, the power is reduced too.

The increased importance of low power design ignited a lot of research in the field. It is known that transformations used at the higher levels of design (like the algorithmic level), achieve the greater power reduction [4]. Algorithmic transformation-based power reduction techniques were presented in Ref. [5]. The idea of re-ordering the sequence of computations of an algorithm, in order to reduce the switching activity in Finite Impulse Response (FIR) filters, was presented in Ref. [6]. Techniques for reduction of power consumption at the inputs of functional units were presented in Ref. [7]. The idea of re-ordering for the minimization of power cost is exploited in Ref. [8], in the transmission of a set of words on a bus, in which the order is not relevant.

The rest of the paper is organized as follows. In the second section, the type of transformational algorithms is discussed. Afterwards, the optimization problem and two algorithms providing solutions to this are presented. In the fourth section, the adaptation of the proposed scheduling algorithms to general implementation architectures is shown. Then, experimental results are presented verifying the power efficiency of the proposed algorithms and finally in the sixth section, conclusions are offered.

TRANSFORMATIONAL ALGORITHMS

This category of algorithms can be represented in tabular form, as it is shown in Eq. (1):

$$\begin{aligned}
 Y = CX \Leftrightarrow & \begin{bmatrix} Y_0 \\ Y_1 \\ \vdots \\ Y_{N-1} \end{bmatrix} \\
 = & \begin{bmatrix} C_{00}, & C_{01}, & \dots, & C_{0(M-1)} \\ C_{10}, & C_{11}, & \dots, & C_{1(M-1)} \\ \vdots & \vdots & \vdots & \vdots \\ C_{(N-1)0}, & C_{(N-1)1}, & \dots, & C_{(N-1)(M-1)} \end{bmatrix} \\
 \times & \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{M-1} \end{bmatrix}
 \end{aligned} \tag{1}$$

Widely used common DSP algorithms, as are DCT and DFT, belong to this class of algorithms.

If the coefficient matrix of the algorithm contains N rows and M columns, the computation of the algorithm involves the calculation of N inner products, each one of these, consisting in M partial products.

The order, by which the partial products of each inner product are calculated, is of no importance, as it is also true for the calculation order of the inner products. This observation is evident from the commutative property of real numbers. The aim is the exploitation of this property, reordering the sequence of computations that constitute the transformational algorithms execution, in order to reduce the switching activity in the address and data buses of the system, implementing the computations. The combination of these observations provides the freedom to re-order first, either the inner products or the partial products.

The above statements guide the derivation of two optimization algorithms, which take as input the coefficient matrix and information concerning the dynamic data and produce specific orders of computation, for the inner and partial products.

PROBLEM FORMULATION

In this paper, techniques for the minimization of switching activity on the data and address buses of the systems, which are implementing transformational DSP algorithms, are presented. This minimization procedure is formulated as a combinatorial optimization problem. The switching activity cost of a bus, for two successive time instances, is modeled efficiently by the Hamming distance between the two successive values of the variables on the bus, at these specific time instances. The cost function that drives the optimization procedure is the total switching activity of the data and address buses. During the calculation of the transformational algorithm, the cost function is computed by the summation of the Hamming distances observed between successive values on the binary words of the data and address buses. The type of information, concerning the above summation is both, static and dynamic. The coefficients of the transformational algorithm, which are stable and known *a priori*, are the static information. The Hamming distance between each pair of coefficients can be calculated easily, given their binary representation. The dynamic information is the data, which the transformational algorithm processes. Information about the dynamic data can be obtained simulating the algorithms using typical data as input. The Hamming distance, between data elements x_k and x_l is observed during the simulation, where k, l are indexes in the M -size one-dimensional vector of Eq. (1). The contents of the data vector are updated according to the algorithm implementation and typical test sequences of image data. The Average Hamming Distance (AHD), in which the Hamming distance between x_k and x_l converges,

characterizes the data, whenever indexes k, l are formed. The notation $SA(x, y)$ is used to indicate the Hamming distance between the x and y values.

The optimization problem for the transformational algorithms can be stated as:

Given the algorithms described by Eq. (1), compute two functions $f(i, j) \in [0, NM - 1]$, $i = 0, \dots, N - 1$, $j = 0, \dots, M - 1$ and $g(i, j) \in [0, M - 1]$, $i = 0, \dots, N - 1$, $j = 0, \dots, M - 1$ such that the new order of computation given by equation:

$$Y_i = \sum_{j=0}^{N-1} C_{f(i,j)} X_{g(i,j)}, \quad \text{for } i = 0, 1, \dots, N - 1 \quad (2)$$

minimizes the cost function given by equation:

$$\begin{aligned} \text{Total } SA(f, g) = & \sum_{i=0}^{N-1} \sum_{j=0}^{N-2} SA(C_{f(i,j)}, C_{f(i,j+1)}) \\ & + SA(X_{g(i,j)}, X_{g(i,j+1)}) \\ & + SA(f(i, j), f(i, j+1)) \\ & + SA(g(i, j), g(i, j+1)) \\ & + \sum_{i=0}^{N-2} SA(C_{f(i,N-1)}, C_{f(i+1,0)}) \\ & + SA(X_{g(i,N-1)}, X_{g(i+1,0)}) \\ & + SA(C_{f(N-1,N-1)}, C_{f(0,0)}) \\ & + SA(X_{g(N-1,N-1)}, X_{g(0,0)}) \\ & + SA(f(N-1, N-1), f(0, 0)) \\ & + SA(g(N-1, N-1), g(0, 0)) \quad (3) \end{aligned}$$

The two re-ordering functions $f(i, j)$ and $g(i, j)$ are two-dimensional with the first variable i pointing to the inner products and the second one j pointing to the partial products. The function $f(i, j)$ takes $N \cdot M$ discrete positive integer values, between 0 and $N \cdot M - 1$, while the $g(i, j)$ takes M discrete positive integer values, between 0 and $M - 1$. Assuming that the storage of the $N \times M$ coefficients is made in a one-dimensional array and the switching activity information concerning the data is stored in a one-dimensional array too, the ordering functions for the original form of the transformations is $f(i, j) = i \cdot M + j$ and $g(i, j) = j$. These values of $f(i, j)$ and $g(i, j)$ functions give the conventional computation order of the inner and partial products, of a matrix-vector multiplication. Although the function g concerns one-dimensional quantity, that is the data vector, it has two arguments for the consistency of notation. The j -th partial product of the i -th inner product is the $C_{f(i,j)} X_{g(i,j)}$ product. Let assume that after the reordering holds $f(i, j) = k$ and $g(i, j) = l$, for specific values of i and j . This means that,

the j -th partial product of the i -th inner product will be formed by the product of the k -th coefficient, (as before it is assumed that the coefficients and data are stored sequentially), and of the l -th data, that is $C_k X_l$.

As it is evident from Eq. (3), the coefficients bus, the data bus, and their corresponding address buses are taken into account in the formulation of the cost function. In Eq. (3), the first double summation accounts for the switching activity of the data and address lines, for the computation of all the inner products. It involves the switching activity of the coefficients bus, the data bus, and their corresponding address buses. The incorporation of the data address bus in the cost function formulation, is made through the indexing of the data bus, while the incorporation of the coefficients address bus through the indexing of the coefficients bus, respectively. The outer summation index, runs through all the inner products. The inner summation index runs through all the partial products of the inner product that is indicated by the specific outer summation index. The second single summation accounts for the switching activity occurred, when the computation proceeds with the next inner product. More specifically, this term accounts for the transition of the calculation from the re-ordered i -th inner product to the re-ordered $(i + 1)$ -th inner product. This transition involves the switching activity occurred as the computation of the transformational algorithm proceeds, from the N -th (last) partial product of the i -th inner product to the first partial product of the $(i + 1)$ -th inner product. As in the previous double summation term accounts for the data bus, the coefficients bus, and the corresponding address buses. Finally, the last four terms are included to account for the transition from the last inner product to the first inner product, assuming an infinite loop of computation, as it is the case for DSP algorithms.

The result of Eq. (3) is the total switching activity occurred at the data and the address buses of the system implementing the DSP transformation, after the sequence of computations has been re-ordered according to the $f(i, j)$ and $g(i, j)$ functions.

The problem can be represented as a hierarchical scheduling problem and its graphical notation is shown in Fig. 1. The input data structure of the problem, which is the coefficient matrix with the information of the AHD concerning the data, is represented by a multi-graph. The main graph is the inner products graph. The nodes of the graph are the inner products of the transformation algorithm. The edges of the graph model the unrestricted transitions from inner product to inner product, during the computation procedure of the transformational algorithm. Thus the graph is complete. In Fig. 1 specifically, the inner products graph consists of four inner products for illustration purposes. The edges of the graph are weighted with the switching activity between the two inner products, which the edge connects. The switching activity term includes the data bus, the coefficients bus, and their corresponding address buses. In fact, the inner products are connected through partial products. The edges of the

FIGURE 1 The multi-graph of the scheduling problem.

inner product graph connect some partial products of the connecting inner products. Suppose that the i_1 -th inner product is connected with the i_2 -th inner product, through the j_1 -th and j_2 -th partial products of i_1 -th and i_2 -th inner products, respectively. Then the cost of the edge that connects the two inner products is given by the following equation:

$$\begin{aligned}
 SA(f(i_1, j_1), g(i_2, j_2)) &= SA((C_{f(i_1, j_1)} X_{g(i_1, j_1)}), \\
 (C_{f(i_2, j_2)} X_{g(i_2, j_2)})) &= SA(C_{f(i_1, j_1)}, C_{f(i_2, j_2)}) \\
 &\quad + AHD(X_{g(i_1, j_1)}, X_{g(i_2, j_2)}) \\
 &\quad + SA(i_1, i_2) + SA(j_1, j_2) \quad (4)
 \end{aligned}$$

Each inner product can be represented as a sub-graph of the main graph. The nodes of the sub-graph are the partial products of the corresponding inner product. Consisting of the partial products of the inner product. The edges of the graph model the unrestricted transitions from partial product to partial product, during the computation procedure of the transformational algorithm. Thus the sub-graph is complete. In Fig. 1 specifically, each inner product consists of four partial products for illustration

purposes. The edges of the sub-graphs are weighted with the switching activity between the partial products, which they connect. The connection cost of the edges is given by Eq. (4), except that in this case the index for the inner products is stable. This is true, because the computation of the transformational algorithm in sub-graphs deals with the calculation of an inner product, passing through its partial products. For the sub-graphs holds:

$$i_1 = i_2 = i \quad \text{and} \quad SA(i_1, i_2) = 0 \quad (5)$$

The solution of the problem is to find a closed path, which passes through every inner and partial product exactly once and has minimum cost, that is the sum of the edge weights, as given by Eq. (3), is minimum. The restriction of passing through every inner product exactly once means that, when the path is inside an inner product sub-graph, passes through every partial product of it exactly once and leaves the inner product node, visiting another, without returning to it again. The bold edges in the inner products graph illustrated in Fig. 1, show a decision for the visiting order of the inner products. Suppose that, it has been decided that the inner products will be visited according to the edges in the list $\{e1, e2, e3, e4\}$. In addition, let the bold edges inside each sub-graph illustrate a desired path, for the visiting

Algorithm 1
Input: The multi-graph $G(V, E)$ consisting of N $IP_i(V_i, E_i), i = 0, \dots, N-1$ graphs
Output: A closed path, visiting each $IP_i(V_i, E_i)$ graph and each partial product of the previous inner product graphs exactly once, with near minimum cost

```

mcc ← ∅;
IP(VIP, EIP) ← ∅;
ip_tour ← ∅;
for j=0 to N-1 open_path[j] ← ∅;

{mcc, IP(VIP, EIP)} = ConstructInnerProductsGraph(G(V, E));

if |VIP| < 9 then ip_tour = exact_tsp(IP(VIP, EIP))
else ip_tour = heuristic_tsp(IP(VIP, EIP));

for j=0 to N-1
    open_path[j] = RestrictedOpenPath ( IPj, mcc );
closed_path = ( ⋃j=0M-1 open_path[j] ) ∪ ip_tour
ConstructInnerProductsGraph( IP0, IP1, ..., IPM-1, mcc )
{
    min = ∞;
    for i = 0, ..., N-1
        for j = 0, ..., N-1
            for k = 0, ..., M-1
                for l = 0, ..., M-1
                    if (k = l) or (i = j) continue;
                    if HD(p(i,k), p(j,l)) < min then
                        {
                            min = HD(p(i,k), p(j,l));
                            mcc[i, j] ← k;
                            mcc[j, i] ← l;
                            IP(VIP, EIP) ← (p(i,k), p(j,l));
                        }
}

```

FIGURE 2 Pseudo-code listing for Algorithm 1.

order of the partial products of the sub-graph. It can be seen that, the closed path formed by the bold edges of the subgraphs in Fig. 1, which are connected with the $e1$, $e2$, $e3$ and $e4$ edges of the main graph, is a candidate solution to the problem. Observe that the inner product IP_0 is connected with the edges $e1$ and $e4$ with the rest of the main graph. The previously mentioned closed path, passes exactly once through every partial product of IP_0 , (namely $PP0_0$, $PP0_1$, $PP0_2$ and $PP0_3$). Enters the IP_0 with the $e4$ edge and leaves it from $e1$ edge. The interchange of edges $e1$ and $e4$ in the previous sentence still is valid, but makes no sense, since the graphs are undirected. Similar reasoning can be stated for every node of the inner products graph. The bold closed path $\{e1, e2, e3, e4\}$ of the inner products graph, guarantees that each inner product will be visited exactly once. If the paths indicated by the bold edges in Fig. 1 are minimum, then the closed path that is formed by them is the solution to the problem.

OPTIMIZATION ALGORITHMS

Two general problem instance cases are identified. Each general case can be regarded as a combinatorial optimization problem, based on the well-known TSP, which is a NP-complete problem. The problem is a restricted form of the TSP. A minimum cost closed path must be found, as the one depicted in Fig. 1, passing from all the partial products of the transformational algorithm. The problem is restricted, because every partial product of an inner product must be ordered, before the sequence of partial products continues with the next inner product. This problem is very difficult to be solved for large problem instances. In Figs. 2 and 3 the pseudo-code listings for the two proposed heuristic algorithms, namely Algorithm 1 and 2 are given. Both algorithms take as input the multi-graph of the problem, with switching activity cost weighted edges, like the one presented in Fig. 1.

Algorithm 2
Input: The multi-graph $G(V, E)$ consisting of N $IP_i(V_i, E_i), i = 0, \dots, N-1$ graphs
Output: A closed path, visiting each $IP_i(V_i, E_i)$ graph and each partial product of the previous inner product graphs exactly once, with near minimum cost

```

for j=0 to M-1
    ordered_paths[j]=OrderPartialProducts( $IP_j$ );
closed_path=ScheduleInnerProducts(ordered_paths);

ScheduleInnerProducts(ordered_paths)
{
    ce  $\leftarrow \emptyset$ ;
    closed_path  $\leftarrow \emptyset$ ;
    for i=0,...,N-1
        for j=0,...,N-1
            min= $\infty$ ;
            if (i=j) continue;
            if HD(ordered_paths[i,0], ordered_paths[j,0]) < min
            {
                min=ordered_paths[i,0];
                ce  $\leftarrow$  (ordered_paths[i, 0]), ordered_paths[j, 0]
            }
            if HD(ordered_paths[i,0], ordered_paths[j,M-1]) < min
            {
                min=ordered_paths[i,0];
                ce  $\leftarrow$  (ordered_paths[i, 0], ordered_paths[j, M-1]);
            }
            if HD(ordered_paths[i,M-1], ordered_paths[j,0]) < min
            {
                min=ordered_paths[i,M-1];
                ce  $\leftarrow$  (ordered_paths[i, M-1], ordered_paths[j, 0]);
            }
            if HD(ordered_paths[i,M-1], ordered_paths[j,M-1]) < min
            {
                min=ordered_paths[i,M-1];
                ce  $\leftarrow$  (ordered_paths[i, M-1], ordered_paths[j, M-1]);
            }
            closed_path  $\leftarrow$  ordered_paths[i]  $\cup$  ordered_paths[j]  $\cup$  ce;
        }
    }
}

```

FIGURE 3 Pseudo-code listing for Algorithm 2.

The algorithm depicted in Fig. 2, calls two procedures. The first procedure is *ConstructInnerProductGraph*, which produces two outputs. These outputs are the graph $IP(V_{ip}, E_{ip})$ and the array mcc . The graph $IP(V_{ip}, E_{ip})$ consists of the nodes set V_{ip} , which are the inner products of the input multi-graph and the edges E_{ip} , which are the minimum cost connection edges, which connect all the pairs of inner products. The procedure finds the minimum cost connection partial product pairs, between all the inner products, using a brute force approach to accomplish this task. The Hamming distances among all the partial products between each pair of inner products are compared. Essentially this step compares the cost of all the possible connection edges, between two inner products. Assuming N inner products, each one consisting of M partial products, the complexity of this procedure is $O(N^2M^2)$. The smallest cost edge is selected for insertion in the graph $IP(V_{ip}, E_{ip})$, for every pair of inner products. In Fig. 2, the notation $p_{(i,k)}$ represents the k -th partial product of the i -th inner product and the pair of partial products $(p_{(i,k)}, p_{(j,l)})$ represents the edge connecting the products. In addition, the minimum cost

connection edge is inserted in array mcc , in the following way. Suppose the comparison between the partial products of i -th inner product against the partial products of all the other inner products. Let the comparison lead to a smallest Hamming distance between the l -th partial product of i -th inner product and the k -th partial product of j -th inner product. Then two entries in array mcc are inserted, that is $mcc[i,j] = l$ and $mcc[j,i] = k$. Recall Fig. 1 and suppose that the procedure *ConstructInnerProductGraph* operates on the multi-graph of it. Assuming that the four edges $e1$, $e2$, $e3$ and $e4$, indicated in Fig. 1, are the minimum cost connection edges between the inner products that they connect, then the output of the procedure will be an array indicating the structure of these four edges. For example, the $mcc(0,1)$ entry of the array will be the $PP0_1$ partial product of inner product $IP0$, because the inner products $IP0$ and $IP1$ are connected through the $PP0_1$ partial product of the $IP0$. Similarly $mcc(1,0) = PP1_0$. These two entries of the mcc array indicate the $(PP0_1, PP1_0)$ pair of partial products, which form the $e1$ edge.

After this procedure the inner products graph has been constructed and the information concerning the

connection edges have been saved in array *mcc*. The remaining problem is to find a minimum cost closed tour, among the cities of the inner product graph $IP(V_{ip}, E_{ip})$ and a minimum cost open path inside each inner product graph, given the connection matrix *mcc*. The minimum cost closed path in $IP(V_{ip}, E_{ip})$ graph is the solution of TSP for this graph. If the cardinality of $V_{ip} \leq 9$ an exact solution of TSP is saved in the set *ip_tour*. This is accomplished by the routine *exact_tsp*, which solves the TSP using a brute force procedure. If the cardinality of $V_{ip} > 9$, then the routine *heuristic_tsp* provides a heuristic solution for TSP in *ip_tour* set, based on genetic algorithms. The listings for these two routines are not provided for economy of space reasons. In fact the *ip_tour* set holds the order of computation of the inner products. The last and first entries of *ip_tour* are the same in order to close the path.

The final step is to find the minimum cost open paths for all the inner product graphs. The N sets *open_path* [j], $j = 0, \dots, N - 1$ are used for the storage of the paths. The paths are restricted, meaning that the starting and ending cities of the path are given. They are the two partial products that participate in the minimum cost connection edges, with which each inner product node is connected with the rest of the inner products graph. Here the usability of *mcc* array is shown more clearly. The terminal nodes of the restricted open path can be found by accessing the *mcc* array. This is a restricted minimum cost open path problem. Its solution is readily available from the solution of the TSP. A TSP problem instance is solved for each inner product graph. The restricted open path results from the solution of TSP, by removing the connection edge between the two terminal partial products. The procedure *RestrictedOpenPath* solves this problem for every inner product graph. It is executed N times, one for every inner product, and each time returns the open path in one of the *open_path* sets. This problem is also an NP-complete problem. For large numbers of partial products ($M > 12$) it is practically unsolvable, but for the practical DSP transformations we encountered in the experimental result section, a brute force approach was possible, finding an exact solution, because $M \leq 8$. For larger problem instances the TSP can be solved by heuristic approaches, which are based on genetic algorithms, on Minimum Spanning Trees (MST), or the Christofides' algorithm [9]. The complexity of calculating the open paths for every inner product graph is $(O(N(M!)))$ if a brute force *RestrictedOpenPath* procedure is used.

The complexity of the Algorithm 1 is dominated by the *RestrictedOpenPath* procedure, if a brute force approach is used, while it is dominated by the *ConstructInnerProductsGraph* procedure if a heuristic for *RestrictedOpenPath* is used.

The algorithm depicted in Fig. 3, calls two procedures too. In this case an opposite approach is followed, than in Algorithm 1. The partial products of each inner product

are ordered first by the function *OrderPartialProducts* and afterwards they are connected by the procedure *ScheduleInnerProducts*, in order to form a closed path with minimum cost, which is the solution to the problem. The procedure *OrderPartialProducts* operates on every inner product sub-graph. Its purpose is to find a minimum cost open path. Once again this is a different form of the TSP problem. Its solution is readily available obtained from the solution of the TSP, discarding the edge with the greater cost. The same observations about heuristic or exact methods for solving the TSP are valid, as in the case of Algorithm 1. Due to the small problem instances encountered in the experimental results section, a brute force exact solution was achieved. Due to lack of space the pseudo-code for the *OrderPartialProducts* procedure is not given. Next, the open paths found by the execution of procedure *OrderPartialProducts* for all inner products are connected calling the procedure *ScheduleInnerProducts*. This procedure compares the costs between all the possible connections of the open paths terminal nodes, and selects the connection with the minimum cost. Four comparisons (that is, every possible combination) are made between the terminal nodes of two ordered paths. The minimum cost connection edge *ce*, consisting of the pair of terminal nodes with the smaller Hamming distance, is inserted in the set *closed_path*, together with the re-ordered open paths that the edge connects. The above comparisons are made for every pair of re-ordered partial product paths and the final solution is formed, inserting in *closed_path* the corresponding paths and connection edges. The complexity of the *ScheduleInnerProducts* procedure is $O(4M^2)$, because for all the combinations of terminal nodes pairs four comparisons are made. The complexity of Algorithm 2 is dominated by the *OrderPartialProducts* procedure if either an exact or a heuristic implementation is attempted. Selecting a solution based on MST for implementing *OrderPartialProducts* requires $O(M^2)$ time. The procedure is called N times for the re-ordering of the partial products in each inner product. Thus, $O(NM^2)$ time is required for the computation of the partial products re-ordering. The complexity of Algorithm 2 is dominated by the *ScheduleInnerProducts* procedure, assuming an MST based heuristic for the implementation of the *OrderPartialProducts* and $N < 4$.

After the final ordering of the partial products, which results from the closed path in the multi-graph of the problem, the forming of functions $f(i, j)$ and $g(i, j)$ is straightforward. Assume that the solution to the problem is formed by the nodes connected with the bold edges shown in Fig. 1. Considering a clockwise visiting direction for the nodes of the path, the ordering of partial products for inner product IP_0 is $\{PP_0_2, PP_0_3, PP_0_0, PP_0_1\}$. Assume also that in the conventional computation order of the transformation algorithm the IP_0 inner product is the first (it has $i = 0$ index) and the partial products $PP_0_0 = C_0X_0$, $PP_0_1 = C_1X_1$, $PP_0_2 = C_2X_2$ and $PP_0_3 = C_3X_3$ are the first, second, third and forth, respectively. Choosing

arbitrarily to start the new sequence of computation from *PPO*₂ partial product the $f(i,j)$ and $g(i,j)$ functions for inner product *IP0* are computed as:

$$\begin{aligned} f(0,0) &= g(0,0) = 2 & f(0,1) &= g(0,1) = 3 \\ f(0,2) &= g(0,2) = 0 & f(0,3) &= g(0,3) = 1 \end{aligned} \quad (6)$$

In Eq. (6), both functions $f(\cdot)$ and $g(\cdot)$ take the same values, because they refer to the first inner product of the transformational algorithm in its original computation sequence.

Consider a transformational algorithm using a two dimensional coefficients array $C_{N \times M}$ of size $N \times M$ and a one dimensional data array X_M of size M , as it is shown in Eq. (1). In general the functions $f(\cdot)$ and $g(\cdot)$ are computed in the following way: The coefficients and data of the transformational algorithm are assigned indexes, according to the following equation:

$$\begin{aligned} C[i,j] &= C_{iN+j}, i = 0, \dots, N-1, j \\ &= 0, \dots, M-1 \quad X[j] = X_j, j = 0, \dots, M-1 \end{aligned} \quad (7)$$

The subscript indexes of coefficients in Eq. (7) are the range of values for the $f(\cdot)$ function, while the subscript indexes for data are the range of values for the $g(\cdot)$ function. The i and j indexes are increased sequentially according to the original computation sequence of the transformational algorithm. Each partial product (node) of the multi-graph of the problem, shown in Fig. 1, is assigned two values. These are the indexes of the coefficient and datum, according to Eq. (7), that participate in the formulation of the partial product.

After the closed path solution provided by Algorithm 1 or 2, a specific partial product is chosen arbitrarily as a starting node for the path. The indexes i and j start to increase automatically from zero, while visiting the nodes of the path in one direction. The first index is increased when a transition to another inner product (sub-graph) is made. The second index is increased when a transition to another node in the same sub-graph is made, while is zeroed again when a transition to another sub-graph is made. The function $f(i,j)$ takes the value of the coefficient index that participates in the formulation of the partial product that is currently met, during the tour of the path. The function $g(i,j)$ takes the value of the data index that participates in the formulation of the partial product that is currently met, during the tour of the path.

The final closed path can be assigned to one or more MACs, according to the target implementation architecture and the design constraints. At the moment, a simple assignment procedure is applied if more than one MAC is required. The ordering is divided into as many parts as the number of MACs and each part is assigned to one MAC. A more efficient assignment procedure is an issue under investigation.

TARGET ARCHITECTURES

The proposed transformations can be used in a low power synthesis environment of transformational algorithms. In addition, the transformations are independent of the implicit architecture of implementation. The proposed cost function can be adapted in order to accommodate varying target architectures. Depending on the target architecture structure, the capacitive loads that the data and address buses demonstrate will be different. This fact can be reflected in the cost function, by weighting the switching activity terms of the corresponding buses with capacitive coefficients. In this way, the realization hardware architecture of the transformational algorithm is being taken into account, resulting in a more realistic cost function. If the capacitive load of the target architectures global buses is known (from a characterization procedure for example), then the different cost functions that are formed, provide the means for design space exploration.

In Eq. (3) only Hamming distance terms are used to model power cost, accounting for the data and address bus switching activity. Each architecture, according to its organisation, presents specific values of capacitance for each switching data or address bus line. A central data or address bus, spanning the system implementing the transformational algorithm will present much more greater capacitance, than a more localized bus. Centralized architectures, with large background memories, have greater capacitance per switching data or address line, than more distributed ones, with many small foreground memories [10].

The capacitance per data or address bit is unknown at the algorithmic level of design flow. These capacitances can be estimated using either previous design experience with various architectures, or using known Register Transfer Level (RTL) capacitance models, presented in the literature [11]. The cost function of Eq. (3) provides switching activity cost. Scaling the Hamming distance terms with corresponding capacitance values, the cost function is transformed to provide power cost estimates, assuming that the operating voltage and frequency are known.

The accuracy of these estimates is increased, because the switching activity on buses is estimated more accurately, since there are not the spurious transitions and glitches encountered in combinatorial logic. The switching activity estimation on the address buses is exact, assuming a specific order of data and coefficient memory accesses, during the transformational algorithm computation. The same is true for the switching activity on the coefficient bus, because the coefficients are known static data. The only source of error is the estimation of the data bus AHD, which has been provided using simulation with typical data. To this error contribute both, the average value estimation of the switching activity and the assumption of some type of typical data. In cases of data with quite different statistics the last error contribution

FIGURE 4 Target architecture with ditributed memories.

will be major. A solution to this problem is the adaptation of the AHD values with new simulations, using the expected data type for the application. Taking into account that this simulation is functional, this task is not time consuming.

In Figs. 4 and 5 two distinct target architectures are shown. The architecture in Fig. 4 is a distributed one, with small local memories for the storage of variables. This

architecture fits well in both custom and programmable hardware. In Fig. 5 a more centralized architecture, common in DSP systems, namely the Harvard architecture, is shown.

If C_{ROM} , C_{A_ROM} , C_{REG} and C_{A_REG} are the switching capacitance in coefficients ROM, its address bus, the register file bus and its address bus, respectively, which are shown in Fig. 4, then the cost function of Eq. (3) is

FIGURE 5 Target architecture with localized large memories.

FIGURE 6 Experimental results in 2's complement, one MAC assignment and sequential storage of coefficients.

modified as:

$$\begin{aligned}
 \text{Total } SA(f, g) = & \sum_{i=0}^{N-1} \sum_{j=0}^{N-2} C_{ROM} SA(C_{f(i,j)}, C_{f(i,j+1)}) \\
 & + C_{REG} SA(X_{g(i,j)}, X_{g(i,j+1)}) \\
 & + C_{A_ROM} SA(f(i,j), f(i,j+1)) \\
 & + C_{A_REG} SA(g(i,j), g(i,j+1)) \\
 & + \sum_{i=0}^{N-2} C_{ROM} SA(C_{f(i,N-1)}, C_{f(i+1,0)}) \\
 & + C_{REG} SA(X_{g(i,N-1)}, X_{g(i+1,0)}) \\
 & + C_{ROM} SA(C_{f(N-1,0)}, C_{f(0,N-1)}) \\
 & + C_{REG} SA(X_{g(N-1,0)}, X_{g(0,N-1)}) \\
 & + C_{A_ROM} SA(f(N-1,0), f(0,N-1)) \\
 & + C_{A_REG} SA(g(N-1,0), g(0,N-1))
 \end{aligned} \quad (8)$$

EXPERIMENTAL RESULTS

The proposed optimization algorithms have been tested with various common used transformational DSP algorithms. The testing procedure was conducted through bit-true simulations written in C language. The DSP algorithms have been tested in both, their original and

transformed form and the switching activity savings have been measured. No special target architecture was taken into account. Assuming an abstract implementation platform and without any information regarding the capacitance of address and data buses, only measures concerning switching activity on the buses were taken.

The simulations were executed in two numerical systems, either in 2's complement or in sign-magnitude, in order to validate our techniques for as many application cases as possible. Images were used as input data for the simulations. Data and addresses were assumed to be 8-bit wide, while the coefficients were fixed-point numbers, with 2-bit wide integer part and F -bit wide fractional part. Three cases have been simulated with $F = 8$, $F = 10$, and $F = 12$, respectively. In addition, results have been taken with the coefficients stored in ROM, either sequentially or reordered, according to the output of the optimization routines. Switching activity measures were obtained, assuming one or two functional units (MACs) implementation.

The lengths of data and address buses are parameters in the simulation environment. Figures 6 and 7 show results in 2's complement arithmetic, using one MAC assignment. The coefficients are stored in ROM either sequentially or re-ordered, according to the optimization algorithm output, respectively.

As it can be seen the best results are observed in the case of 1-D 8pt brute force DFT, where more than 70% in Fig. 6 and 80% in Fig. 7, power savings are obtained. This is due to the coefficient values of the DFT. The achievable savings are significant in every case and with both optimization algorithms. It is evident that there is no significant change in switching activity savings, as the bit-width of the coefficients varies. In Fig. 7, where the

FIGURE 7 Experimental results in 2's complement, one MAC assignment and re-ordered storage of coefficients.

coefficients are stored re-ordered in ROM there is an improvement in power savings. This improvement is about 3% that is quite small, for the 8 and 16pt fast DCTs and can reach about 10% for the brute force 16pt DFT and the Swift and PTL FFTs cases. Storing the coefficients in ROM according to the new optimized order results in a sequential access of them, during the computation of the transformational algorithm. Nevertheless, the data have to be accessed conforming to the new order. Generally, an address penalty (in terms of switching activity) offsets the

benefits of the reordered coefficient storage. In our case this penalty is small, since the address bus for data it is assumed to be 8-bit wide. This is true when the data are stored in small foreground memories (register files). In centralized types of architectures the address bus for data can be quite large and the “scrambling” of data addresses would give inferior results. Nevertheless, a substantial improvement should always be expected, since the address bus is taken into account in the optimization algorithms. The above discussion imposes a relative independence of

FIGURE 8 Experimental results in 2's complement, two MACs assignment and sequential storage of coefficients.

FIGURE 9 Experimental results in 2's complement, two MACs assignment and re-ordered storage of coefficients.

the proposed techniques from bit-width of the coefficients, while the effect of re-ordering of the coefficients in memory can be quite significant in some cases.

Figures 8 and 9 show the same results, assuming assignment in two MACs.

As it can be seen, the results are similar with the one MAC assignment case. There is no significant change in switching activity savings. This is not surprising, since the assignment in two MACs is a straightforward extension of the one MAC case.

In Figs. 10–13 the same type of results are presented, but in sign-magnitude arithmetic.

Comparable performance in terms of switching activity savings it is shown between 2's complement and sign-magnitude arithmetic. In both arithmetic systems the same observations are valid.

The presented results show a comparable performance for both optimization algorithms for the fast DCTs and the brute force 8pt DFT cases. The PTL and Swift FFT algorithms, exhibits a little better performance for the case

FIGURE 10 Experimental results in sign magnitude, one MAC assignment and sequential storage of coefficients.

FIGURE 11 Experimental results in sign magnitude, one MAC assignment and re-ordered storage of coefficients.

of one MAC implementation, realizing the re-ordering according to Algorithm 2, while for two MACs the results are comparable.

Given the new sequence of computation of the transformational algorithms, according to the output order of Algorithm 1 or 2, these transformations does not impose any performance penalty, since they do not increase the computation cycles. For programmable architectures they are easy to be realized. The program that executes the transformational algorithm just has to consider the re-ordering of computation, accessing both coefficients and data in agreement with the new order. In

custom non-programmable architectures the re-ordering of computations can be implemented using a suitable control unit. In this case it is difficult to take into account many different types of data, where we have different AHD values. Although this is a seldom case, since usually the data types processed by the transformational algorithms have similar characteristics, substantial power savings can be achieved re-ordering the computation only according to the coefficients [5,6] and ignoring the effects of data. In this way only a special control unit is needed. Nevertheless, two or more types of data, with AHD that cause different computation orders, can be considered in

FIGURE 12 Experimental results in sign magnitude, two MACs assignment and sequential storage of coefficients.

FIGURE 13 Experimental results in sign magnitude, two MACs assignment and re-ordered storage of coefficients.

custom architectures, realizing corresponding control units, for every type of data. How many control units can be implemented is a trade off between chip area, increased design complexity, and available control pins. The increase in power due to the additional control units is not a problem, since the control units that are not used could be powered down. This is a not difficult implemented power management mean.

CONCLUSIONS

In this paper two power efficient hierarchical scheduling algorithms for DSP transformations have been presented. The algorithms reorder the traditional sequence of computations in DSP transformations, minimizing the Hamming distance that is observed in the data and address buses of the system, implementing the algorithms. Experimental results that have been obtained through simulations of the targeted DSP algorithms with actual image data show that there is a significant reduction of the switching activity in the system's buses. An average value of 30% in switching activity is obtained, while the savings can be up to 70% or more in the case of brute force DFT.

The two proposed algorithms present similar performance in terms of switching activity reduction. Algorithm 2 presents a slightly improved performance (about 10% more reduction in switching activity), for the PTL and Swift FFT algorithms in the case of one MAC assignment of computation.

A significant feature of the proposed scheduling algorithms is their independence from the target architecture. With a suitable modification of the problem cost function, the scheduling can be adapted to accommodate

every DSP architecture. This characteristic of the proposed scheduling techniques makes them attractive for synthesis environments and design space exploration purposes. Another important feature of the proposed scheduling algorithms is that they do not increase the necessary computation cycles, in comparison with the original computation order. They can be implemented easily in programmable architectures, while the effect of data is more difficult to be taken into account in custom architectures. In any case, the cost of simulation with typical data cannot be avoided, although it is not very time consuming, since the simulation is functional.

References

- [1] Sindh, D., Rabaey, J.M., Pedram, M., Catthoor, F., Rajgopal, S., Sehgal, N. and Mozdzen, T.J. (1995) "Power conscious CAD tools and methodologies: a perspective", *Proceedings of IEEE Journal* **83**(4), 570–594.
- [2] Weste, N. and Eshragian, K. (1993) *Principles of CMOS VLSI Design: A Systems Perspective*, 2nd Ed. (Addison-Wesley, Reading, MA).
- [3] Stan, M.R. and Burleson, W.P. (1995) "Bus-invert coding for low-power I/O", *IEEE Transactions on VLSI Systems* **3**, 49–58.
- [4] Rabaey, J.M. and Pedram, M. (1995) *Low Power Design Methodologies* (Kluwer Academic Publishers, Dordrecht).
- [5] Chandrakasan, A.P., Potkonjak, M., Mehra, R., Rabaey, J. and Brodersen, R.W. (1995) "Optimizing power using transformations", *IEEE Transactions on CAD of Integrated Circuits and Systems Journal* **14**(1), 12–30.
- [6] Merakos, P.K., Masselos, K., Koufopavlou, O., Nikolaidis, S., Goutis, C.E. and Novel, A. (1997) "Transformation for reduction of switching activity in FIR filters implementation", *Proceedings of the IEEE International Conference on Digital Signal Processing (DSP)* **2**, 653–656.
- [7] Masselos, K., Merakos, P., Stouraitis, T. and Goutis, C.E. (1999) "Novel techniques for bus power consumption reduction in realizations of sum-of-product computation", *IEEE Transactions on Very Large Scale of Integration (VLSI) Systems Journal* **7**(4), 492–497.

- [8] Murgai, Rajeev, Fujita, Masahiro and Oliveira, Arlindo (1998) "Complementation and resequencing to minimize transitions", *Proceedings of the 1998 ACM/IEEE Design Automation Conference (DAC)*, San Francisco CA, 694–697.
- [9] Steiglitz, Papadimitriou K. (1982) *Combinatorial Optimization: Algorithms and Complexity* (Prentice-Hall, Englewood Cliffs, NJ).
- [10] Diguët, J.P., Wuytack, S., Catthoor, F. and Deman, H. (1997) "Hierarchy exploration in high level memory management", *Proceedings of the International Symposium on Low Power Electronics and Design*, Monterey CA, August 18–20.
- [11] Landman, P.E. and Rabaey, J.M. (1996) "Activity-sensitive architectural power analysis", *IEEE Transactions on Computer-Aided Design* June, 571–587.

Authors' Biographies

Panagiotis Merakos was born in Korydallos of Attiki, Greece, in 1968. He received the Diploma in Electrical Engineering from the University of Patras, Greece in 1992. From 1993 until 1997 was a researcher in the VLSI Design Laboratory of the University of Patras. From 1997 until 1998 he joined the Greek army, where he was trained as a technician in wireless high power communications. After the army he was associated with Atmel Hellas S.A. company, where he involved with the design of communication systems. He is currently a researcher and studies towards a PhD degree at the Department of Electrical and Computer Engineering, University of Patras. His research interests include high-level algorithms and techniques for low-power DSP systems design, analysis and synthesis.

Konstantinos Masselos was born in 1971 in Athens, Greece. He received a first degree in Electrical Engineering from University of Patras, Greece in 1994 and an MSc degree in VLSI Systems Engineering from University of Manchester Institute of Science and Technology (UMIST), United Kingdom in 1996. For his

MSc studies he was awarded a UMIST graduate research scholarship. In March 2000, he got a PhD degree in Electrical and Computer Engineering from University of Patras, Greece.

His PhD research was related to high level low power design methodologies for multimedia applications realized on different architectural platforms. From 1997 until 1999 he was associated as a visiting researcher, with the System Exploration for Memory and Power (SEMP) group of the Design technology for integrated Information and Communication Systems (DESICS) division of the Inter-university Micro Electronics Centre (IMEC) in Leuven, Belgium, where he was involved in research related to the ACROPOLIS multimedia pre-compiler.

His main research interests include low power design, data storage and transfer optimization, design of low complexity algorithms for image and video processing, high level synthesis and compilation techniques. He is a member of the IEEE and of the Technical Chamber of Greece.

Dr Costas Goutis was a Research Assistant and Research Fellow in the Department of Electrical and Electronic Engineering at the University of Strathclyde, U.K., from 1976 to 1979, and Lecturer in the Department of Electrical and Electronic Engineering at the University of Newcastle upon Tyne, U.K., from 1979 to 1985. Since 1985 he has been Associate Professor and Full Professor in the Department of Electrical and Computer Engineering, University of Patras, Greece. His recent research interests focus on VLSI Circuit Design, Low Power VLSI Design, Systems Design, Analysis and Design of Systems for Signal Processing and Telecommunications. He has been awarded a large number of Research Contracts from ESPRIT, RACE and National Programs.