LPGD

A Low-Power Design Methodology/Flow and its Application to the Implementation of a DCS1800-GSM/DECT Modulator/Demodulator

(ESPRIT 25256)

Title: Low Power Methodology for Designing Multiple Filters
Author(s): K. Masselos¹, C. E. Goutis¹, S. Nikolaidis², E. Karaolis², and E. D. Kyriakis-Bitzaros³
¹VLSI Design Laboratory, Department of Electrical and Computer Engineering, University of Patras, Greece.
²Section of Electronics & Computers, Department of Physics, Aristotle University of Thessaloniki, Greece.
³Institute of Microelectronics, NSRC "Demokritos", Greece.

Editor: VLSI Design Laboratory, UP.
Type: Report
LPGD Id: LPGD/ WP3/ UP/D3.2R1
CEC Identifier: EP25256/UP / D3.2R1
Document Version: 1
Status: Report
Confidentiality: Public
Actual Date: November 30, 1998
Contractual Date: December 17, 1998
WorkPackage: WP3
Keywords: Low Power, Switching Activity, Architectural Synthesis, Sum-of-Product Computation, Estimation, Statistics.

Abstract: Techniques for power optimization and estimation of sum-of-product computational kernels at the architecture level are described. Synthesis techniques that reduce the switching activity in the inputs of the functional units are proposed. Information related to both coefficients and input data is included in the cost function that drives the synthesis procedure. Novel heuristics based on the Travelling Salesman’s Problem and the Minimal Spanning Trees are used for scheduling and assignment. A novel method for the accurate calculation of the transition activity in the input and output nodes of single functional unit implementations of FIR filters is also introduced. The method is developed for input signals which can be described by a Gaussian process. The DBT model is used for
modeling the bit-level transition activity of a signal word. The transition activity is calculated as a function of the signal statistics.
Recent advances in the areas of wireless communications made available a large number of portable battery-operated systems such as cellular phones, pagers, wireless modems and portable videophones. Such systems make extensive use of digital signal processing. Since power consumption is the overriding issue in the design of portable systems, the development of design techniques for low power implementation of digital signal processing algorithms is of utmost importance.

Architectural synthesis is used to map algorithms onto hardware structures. Low power architectural synthesis techniques are very important since they lead to the most significant power savings in comparison to those achieved by optimizations performed in the lower levels of abstraction. Power estimation techniques in the architectural level are also important since they can be used for the fast power evaluation of different candidate architectures allowing early architectural decisions.

The purpose of this report is two-fold: In the first step it describes novel techniques for low power architectural synthesis of FIR filters and Sum-of-Product computational structures in general. In the second step it presents novel techniques for the accurate estimation of the switching activity in the input and output nodes of single functional unit realizations of FIR filters.
TABLE OF CONTENTS

1. Power Optimizing Synthesis Techniques for Sum-Of-Product Computational Kernels 6

1.1 Introduction 6

1.2 Related Work 7
1.2.1 Low power architectural synthesis techniques 7
1.2.2 Low power design techniques for digital signal processing 7

1.3 Classification of Algorithms Requiring Sum-of-Product Computation 7

1.4 Target Architecture Model 8

1.5 Proposed Synthesis Techniques for Single Functional Unit Implementations 9
1.5.1 General 9
1.5.2 Convolutional algorithms 10
1.5.3 Transformational algorithms 11

1.6 Experimental Results for Single Functional Unit Implementations 13
1.6.1 FIR filters 13
1.6.2 Two dimensional wavelet transforms 14
1.6.3 Fourier transform 14
1.6.4 Discrete cosine transform 15
1.6.5 Comparison of cost functions 16

1.7 Proposed Synthesis Techniques for Multiple Functional Units Implementations 17
1.7.1 Multiple functional unit architectures 17
1.7.2 Convolutional algorithms 18
1.7.3 Transformational algorithms 19

1.8 Experimental Results for Multiple Functional Units Implementations 20

1.9 Evaluation of the Effects of the Proposed Synthesis Techniques 21

1.10 Comparison with Other Techniques for Low Power DSP 22

2. Techniques for Switching Activity Estimation in FIR Filters Realizations 23

2.1 Introduction 23

2.2 Preliminaries 23

2.3 Propagation of the signal statistics in FIR structures 24

2.4 Mapping the FIR filter structure on a MAC 25

2.5 Signal statistics at the nodes of the MAC 28

2.6 Calculation of the transition activity 30

2.7 Conclusions 34
TABLE OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Target architecture model</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Target architecture model for transformational algorithms and the parallel output computation case</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>Input switching activity savings for the sequential output computation case</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>Input switching activity savings for the parallel output computation case</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>Comparison of the results produced by the different cost functions</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>Centralized memory and bus organization</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>Localized memory and bus organization</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>MAC-based configuration</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>Multiplier-based configuration</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>FIR filter structure</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>MAC unit</td>
<td>26</td>
</tr>
<tr>
<td>12</td>
<td>Actual and theoretic probability density functions at the output of the multiplexer</td>
<td>27</td>
</tr>
<tr>
<td>13</td>
<td>Actual and calculated transition activity per bit for SIG2 and SIG4 at node #3</td>
<td>32</td>
</tr>
<tr>
<td>14</td>
<td>Measured and estimated transition activity per bit for SIG2 and SIG4 at node #4</td>
<td>32</td>
</tr>
</tbody>
</table>
1. Power Optimizing Synthesis Techniques for Sum-Of-Product Computational Kernels

1.1 Introduction

Sum-of-product computation between data and coefficient vectors is the dominant type of computation in many digital signal-processing algorithms like FIR filtering. Novel techniques for low-power synthesis of sum-of-product computation are developed. The proposed techniques aim at reducing the switching activity (equivalent to power consumption), required for the successive evaluation of the partial products, in the busses connecting the storage elements, where data and coefficients are stored, to the functional units (inputs of the functional units). This is achieved through reordering the sequence of evaluation of the partial products.

Two different categories of algorithms requiring sum-of-product computation are identified with respect to their computational structures. In the first category the basic computation is the vector-vector multiplication while the algorithms of the basic computation of the second category is the matrix-vector multiplication. Heuristics based on the Travelling Salesman Problem and the Minimal Spanning Trees are used to find a reduced bus switching activity (power efficient) ordering of the partial products. The cost function that drives the reordering includes information related to both data (dynamic) and coefficients (static). However only the static (coefficient related) information can be used to drive the exploration for the reduced switching activity ordering to speed-up the reordering procedure.

When a sum-of-product computation is mapped on a single functional unit only a reduced bus switching activity schedule needs to be derived. In cases where multiple functional units are present heuristics are used to assign the partial products of the computation to the functional units, in a first step. These heuristics increase the correlation of the partial products that will be assigned to the same functional unit thus reducing the switching activity. Next, the same scheduling techniques as for the single functional unit case, are used to reduce the switching activity at the inputs of the functional units.

The proposed synthesis techniques target customized architectures with register file storage of data to allow exploitation of the data re-use present in sum-of-product computation. The application of the proposed techniques in a programmable processor context is also explored.
1.2 Related Work

1.2.1 Low power architectural synthesis techniques

Power optimization in the architectural level is very important since it leads to much larger savings in comparison to those achieved at the lower levels of abstraction (e.g. the logic/gate level) [1]. Thus the development of architectural synthesis techniques targeting low power consumption is of great significance. A large number of custom data path synthesis techniques for power optimization have been proposed. Transformation-based data path synthesis techniques for low power have been addressed in [2-6]. Scheduling techniques for low power have been presented in [7-11]. In [7-10] scheduling techniques using multiple supply voltages are described while in [11] a scheduling technique enabling power management is described. A technique for simultaneous scheduling and allocation is described in [12] while a simultaneous scheduling and assignment technique is described in [13]. Allocation techniques are described in [14-15] while a technique for assigning operations on functional units is described in [16]. A technique for module selection during architectural synthesis is presented in [17]. An algorithm solving simultaneously the scheduling, allocation, assignment, module selection and clock selection problems are described in [18]. Synthesis techniques exploiting structural properties of the algorithm to be realized in hardware are described in [19-22]. A technique for energy efficient register allocation is described in [23]. Controller-based synthesis techniques are described in [24, 25]. Clock-based power management techniques are proposed in [26, 27]. Synthesis techniques for reducing the switching activity of functional units is described in [28] while a synthesis technique that increase data correlation to reduce switching activity is described in [29]. A profile based synthesis technique starting from VHDL descriptions is described in [30]. Environments for low power architectural synthesis are described in [31-34]. The above mentioned techniques may target digital signal processing applications however they are relatively general and they do not take strongly into consideration the special characteristics of these applications.

1.2.2 Low power design techniques for digital signal processing

Some data path synthesis techniques taking into consideration special characteristics of digital signal processing applications have been proposed in [35-40]. In [41] the problem of algorithm selection for application specific digital signal processing systems is formulated as an optimization problem. A large number of other techniques and methodologies for low power/energy digital signal processing have been proposed. A data driven approach has been proposed in [42] while the use of dynamically adjustable power supplies in a programmable processor context has been proposed in [43]. Algorithm based approaches are described in [44] while transform coding architectures based on multirate approach are described in [45]. Low power algorithms and architectures are presented in [46] while a transformation based technique for low power adaptive signal processing is described in [47]. Techniques for low power/energy digital signal processing in a programmable context are described in [48, 49]. Techniques for low power filtering are presented in [50-62]. Several techniques for power optimization of FIR filters have been proposed in [50-57]. In [50, 51] the transformation of the binary representation of the filter coefficients is proposed while the technique described in [52] perturbs the filter coefficients to decrease the number of operations required to obtain the filter output while preserving the desired filter characteristics. In [53] a technique for low power realization of FIR filters using differential coefficients is presented while in [54] a transformation for power reduction, through minimization of the number of multiplications, in FIR filters is described. Approaches for low power realization of FIR filters in a programmable processor context are described in [55-57]. Techniques for low power adaptive filtering are described in [60-62]. In [63-65] low power architectures for the Discrete Cosine Transform (DCT) are presented while in [66-68] power and energy efficient architectures for the DFT/FFT are described.

1.3 Classification of Algorithms Requiring Sum-of-Product Computation

Digital signal processing algorithms requiring sum-of-product computation have been divided in the following two categories:

a) Convolutional algorithms
The basic computation of the algorithms of this category is the convolution between data and coefficient vectors and is described by eqn. 1. A typical algorithm belonging to this category is the FIR filtering.

\[ Y_n = \sum_{i=0}^{N-1} C_i X_{n-i} \quad (1) \]

where \( C_i \)s are the coefficients of the filter and \( X_i \)s are the \( n_b \) terms of the input and output sequences respectively. The main characteristic of the basic computational structure of the algorithms of this category is that for the evaluation of the output terms the same coefficient vector and different data vectors (overlapping by \( N-1 \) terms and one different term for successive output terms) are used.

b) Transformational algorithms

The basic computation of these algorithms is the matrix-vector multiplication between a coefficient matrix and a data vector and is described by eqn. 2. Typical examples of algorithms belonging to this category are the common DSP transformations (like DCT and DFT-FFT).

\[ Y = CX \Leftrightarrow \begin{bmatrix} Y_0 \\ Y_1 \\ \vdots \\ Y_{N-1} \end{bmatrix} = \begin{bmatrix} C_{00}, & C_{01}, & \ldots, & C_{0M-1} \\ C_{10}, & C_{11}, & \ldots, & C_{1M-1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{(N-1)0}, & C_{(N-1)1}, & \ldots, & C_{(N-1)(M-1)} \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{M-1} \end{bmatrix} \quad (2) \]

where \( Y \) is the \( N \)-point output data \((Y_0 \rightarrow \cdots \rightarrow Y_{N-1})\) vector, \( C \) is the \( N \times M \) coefficient \((C_{i,j})\) matrix, and \( X \) is the \( N \)-point input data \((X_0 \rightarrow \cdots \rightarrow X_{M-1})\) vector. The main characteristic of the basic computational structure of the algorithms of this category is that for the evaluation of the \( N \) output terms different coefficient vectors are used. On the other hand the same data vector is used for all the \( N \) output points of the basic computation. For the algorithms of this category there are two options: I) Sequential output computation and II) Parallel output computation.

1.4 Target Architecture Model

There are many ways of realizing sum-of-product computational structures in hardware as part of digital signal processing systems. One way is to use dedicated hardware multipliers and adders (pure custom hardware approach). A second alternative is the use of an instruction set processor (either a general purpose processor or a digital signal processor) and its computational units (usually multiply/accumulators). Finally a multiply/accumulator based custom hardware architecture allowing sharing of hardware units and offering a low-level programmability and flexibility can be used.

The general view of the target architecture model is shown in figure 1. This architecture can be used for both convolutional kernels and sequential output computation transformational kernels.

Data are transferred to the foreground registers either from a background memory (in case of data dominated applications like image and video processing) or from the circuit inputs (probably output of an A/D converter). The foreground registers store the data terms required for the computation of each output term. In the case of original background data storage the use of registers allows exploitation of the data re-use present in both convolutional and transformational basic computations, favouring power consumption reduction [69]. Specifically for the convolutional algorithms, the use of \( N \) foreground registers to store the input data terms reduces the number of accesses to the background memory from \( N \) to 1 for each output point. For the transformational kernels and the sequential output computation case the use of \( M \) foreground registers to store the input data terms reduces the number of accesses to the background memory from \( N \times M \) to \( M \) for a basic computation. One register file is required for each basic computation performed in parallel. The coefficients are usually stored in a separate memory. In most cases the computations are performed in a multiply/accumulator based (sometimes multiplier-based) data path. The number of functional units used for the computation is determined by the performance requirements of the application. Multiple functional units can be used in two ways: a) To evaluate different basic computations in parallel. b) To compute parts of the same basic computation in parallel. The target architecture for the transformational algorithms and the parallel output computation case is similar except from the fact that the outputs are stored in the registers. The modified target architecture for this case is shown in figure 2.

The proposed synthesis techniques change the order in which data and coefficients are transferred to the functional units (as compared to the original case). This does not introduce any addressing penalty in the
target custom hardware realizations. Data are transferred to the registers in the same order as in the original case. The registers are accessed by signals generated by the control unit and thus no address generation penalty is introduced. As far as the activity of the address lines is concerned, activity reduction techniques can be used as in the original case. The coefficients can be arranged in the memory according to the new order of access.

![Fig. 1: Target architecture model](image1)

![Fig. 2: Target architecture model for transformational algorithms and the parallel output computation case](image2)
1.5 Proposed Synthesis Techniques for Single Functional Unit Implementations

1.5.1 General

The proposed techniques reorder the sequence of evaluation of the partial products that constitute the basic computations. The aim of the reordering is the minimization of the sum of Hamming distances between successive pairs of partial products (switching activity), in the busses connecting the data and coefficient storage elements to the functional units. This is equivalent to bus power consumption reduction. It is assumed that for each functional unit used for the computation there are two different busses one for data and one for coefficients connecting the corresponding storage elements to the functional unit. The term $HD(c_i d_i, c_j d_j)$ given by eqn. 3, will be mentioned as the Hamming distance between the partial products $c_i d_i$ and $c_j d_j$ and represents the total Hamming distance in the busses when the partial products $c_i d_i$ and $c_j d_j$ are successively evaluated on the same functional unit.

$$HD(c_i d_i, c_j d_j) = HD(c_i, c_j) + AHD(d_i, d_j) \quad (3)$$

The first term on the right hand side of eqn. 3 represents the Hamming distance between the coefficients of the partial products and can be directly evaluated since coefficients are known before realization. The second term on the right hand side of eqn. 3 is the Average Hamming Distance (AHD) between the data terms of the partial products and is determined through simulation. The evaluation of the exact Hamming distance between two data samples is not possible since data are not known before run-time. An approximation (AHD) can be obtained by simulation of typical data [70] of a specific application (e.g. test images for image processing applications or test sequences for video processing applications). For every possible pair of samples $d_i, d_j$ that belong to the data vector of a basic computation, the Average Hamming Distance ($AHD_{ij}$) is evaluated. The simulation stops when a convergence criterion (described by eqn. 4) is satisfied:

$$\frac{AHD_{ij}(M) - AHD_{ij}(M + 1)}{AHD_{ij}(M)} < \text{Threshold} \quad \text{for all } i, j \quad (4)$$

Where $AHD_{ij}(M)$ is the Average Hamming Distance between the $i_{th}$ and $j_{th}$ data terms of the data vector after $M$ simulation cycles.

In this way information related to both data and coefficients is used to drive the reordering of computation. However only the static information can also be used to determine the final order of the computation of the partial products. In such a case the quality (in terms of switching activity reduction) of the derived order is traded-off for an increased speed of the reordering procedure.

In this section for simplification it is assumed that the basic computations (as defined in section 2) are performed on a single functional unit (multiply/accumulator). The synthesis techniques for the single functional unit case will form the basis for the development of techniques for multiple functional unit implementations.

1.5.2 Convolutional algorithms

The convolution operation performed by the algorithms of this category for the evaluation of an output term is described by the following equation:

$$y_n = \sum_{f(k)=0}^{N-1} C_{f(k)} X_{n-f(k)} \quad (5)$$

where $y_n$ is the $n_{th}$ term of the output sequence, $C$ is the coefficient vector and $X$ is the vector of the input data. The computation is performed according to the ordering function $f(k)$, $k=0,\ldots,N-1$. The ordering function generates for each partial product a relative order of computation and it is one-by-one and onto. In the original case $f(k)=k$.

The total Hamming distance of the sequence of evaluation of the partial products on a single functional unit as determined by the ordering function $f(k)$ is given by the following equation:

$$\text{Total } HD(f(k)) = \sum_{k=0}^{N-1} HD(p_{f(k)}, p_{f(k)+1}) + HD(p_0, p_{N-1}) \quad (6)$$
where \( p_{\text{sk}} \) is the partial product \( c_{\text{sk}}x_{\text{sk}} \). The term \( \text{HD}(p_0, p_{N-1}) \) denotes the Hamming distance between the first \( (f(k)=0) \) and the last \( (f(k)=N-1) \) partial product (according to the ordering function \( f(k) \)). This term is included because after the computation of the last partial product \( (p_{N-1}) \) for the output term \( y_n \) the first partial product \( (p_0) \) for the term \( y_{n+1} \) must be computed. The Hamming distance for a pair of partial products \( \text{HD}(p_k, p_l) \) is given by eqn. 3. The aim of the proposed techniques is to derive the ordering function \( f(k) \) that minimize the total Hamming distance of the convolutional computation as given by equation 6. Since there are no data dependencies between partial products and taking into consideration that the basic computation is performed on a single multiply-accumulator, the problem of finding the minimum cost (in terms of total bus switching activity) ordering function is equivalent to finding the minimum cost schedule of the partial products on the multiply/accumulator.

The problem of computation reordering for the convolutional algorithms is formulated as a Travelling Salesman Problem (TSP). The graph \( G(V, E) \) of the problem consists of the set \( V \) of the \( N \) vertices, which are the partial products required for the computation of an output term, and the set \( E \) of edges which model the unconstrained transition from one partial product to another. The problem’s graph is complete i.e. each vertex pair is connected by an edge. To each edge of the graph a weight, which is the Hamming distance between the two partial products that the edge connects as defined by eqn. 3 is assigned. A closed path over all vertices (partial products) must be found, without passing from a vertex more than once, resulting in a minimum cost. The weights of the edges between the vertices are real positive values. Their lower bound is zero and their upper bound equals to the sum of the number of bits used for the representation of the coefficients and the representation of data terms. If the simpler cost function (including only the static information related to coefficients) is used the costs of the edges are bounded positive integer numbers. Their lower bound is zero and the upper bound equals to the number of bits used for the representation of the coefficients.

Several algorithms have been proposed for the solution of the TSP problem. If the size of the problem is relatively small, an exact solution can be found in short time. A large number of exact algorithms have been proposed for TSP which can be best understood and explained in the context of Integer Linear Programming (ILP) [71]. The NP-complete class of the problem motivated the research for heuristic algorithms. Christofides in [72] proposed a heuristic using a minimum-cost matching algorithm and requiring \( O(n^2) \) time to find a nearly optimal solution.

1.5.3 Transformational algorithms

Sequential output computation

The sequential computation of the output points of a transformation of length \( N \times M \) is described by the following equation

\[
y_f(i) = \sum_{g(i,j)=0}^{M-1} c_f(i,g(i,j))x_{g(i,j)}, \quad \text{for } f(i) = 0,1,........N-1
\]

where \( y \)'s are the output points, \( c \)'s are the \( M \) coefficients of one row of the coefficient matrix and \( x \)'s are the \( M \) elements of the input data vector. Since the coefficient structure is two-dimensional two ordering functions are required to describe the basic computation in this case.

The computation is performed according to the ordering functions \( f(i), g(i,j), i=0,1,2,....N-1, j=0,1,2,....M-1 \). The ordering functions \( f(i), g(i,j) \) are one-by-one and onto and in the original case \( f(i)=i \) and \( g(i,j)=j \). The \( f(i) \) ordering function corresponds to the rows of the coefficient matrix and the \( g(i,j) \) ordering functions (one per row of the coefficient matrix) correspond to the columns of the coefficient matrix. The \( f(i) \) function determines the order in which the inner products are computed (order in which output terms are evaluated) while the \( g(i,j) \) functions determine the order in which the partial products that form an inner product are computed and are different for each inner product. The total Hamming distance of the sequence of evaluation of the partial products that constitute the basic computation as determined by the ordering functions \( f(i), g(i,j) \) on a single functional unit, is given by the following equation:

\[
\text{Total } \text{HD} \left( f, g \right) = \sum_{n=1}^{N-1} \sum_{k=1}^{M-1} \text{HD} \left( p_{f(i),g(i,j)}, p_{f(i),g(i,j)+1} \right) + \text{HD} \left( p_{f(i),N-1}, p_{f(i)+1,0} \right)
\]
where \( p_{(i,j)} \) is the partial product \( c_{(i,j)} x_{(i,j)} \). The Hamming distance for a pair of partial products is given by eqn. 3. The term \( HD(p_{(0,N-1)}, p_{(0,1,0)}) \) denotes the Hamming distance between the last partial product of the inner product \( f(i) \) and the first partial product of the inner product \( f(i)+1 \). This term is included because after the computation of the last partial product \( f(i) \) for the output term \( y_{(0)} \) the first partial product \( p_{(0,1,0)} \) must be computed.

The proposed techniques aim at deriving the ordering functions \( f(i), g(i,j) \), that minimize the total Hamming distance of the basic computation as described in equation 8. There are no data dependencies between inner products and partial products belonging to the same inner product. Thus the problem of deriving minimum switching cost ordering functions for the inner and the partial products of the basic computation is equivalent to deriving minimum switching cost schedules for the inner products and the partial products on the single multiply/accumulator.

The formulation of the computation reordering problem is more complex for the transformational algorithms. In a first step the minimum cost ordering of the inner products IP, of the basic computation must be determined. In the second step the minimum cost ordering of the partial products \( p_k \) (\( j=0,1,\ldots,M-1 \)) that constitute each inner product, must be found. The problem is tackled in the following way: For all possible pairs of inner products \( IP_i, IP_j \), \( i=0,1,\ldots,N-1, j=0,1,\ldots,N-1 \) the minimum cost (switching activity) connection is determined. The minimum cost connection is defined as the pair of partial products \( p_k \) (belonging to the inner product \( IP_k \) \( k=0,1,\ldots,N-1 \)) and \( p_l \) (belonging to the inner product \( IP_l \) \( l=0,1,\ldots,N-1 \)) that minimizes the Hamming distance between all possible pairs of partial products as described by equation 9. The complexity of this procedure is \( M^2 N (N-1) \).

Minimum cost connection for \( IP_i \) \( IP_j : (p_{ik} , p_{jl}) \)

\[
HD(p_{ik} , p_{jl}) \leq HD(p_{im} , p_{jn}) \quad \text{for all } m,n, \quad m,n=0,1,\ldots,M-1 \quad (9)
\]

Since the minimum cost connection has been determined for all possible pairs of inner products the graph \( G(V,E) \) of the problem can be constructed where \( V \) is the set of \( N \) vertices while \( E \) is the set of edges. The graph is complete and the \( N \) vertices correspond to the \( N \) inner products while edges model the unconstrained transitions from one inner product to another. To each edge the minimum cost connection corresponding to the inner products connected by the edge is assigned as a cost. The determination of a minimum cost ordering of the inner products is formulated as a Travelling Salesman Problem (TSP) on this graph. A closed path over all vertices (inner products) must be found, without passing from a vertex more than once, resulting in a minimum cost. For the solution of the TSP the strategies proposed for the convolutional algorithms can be used. As soon as the minimum switching ordering of the inner products is determined, the partial products that constitute each inner product must be ordered to minimize the switching required for its computation. In fact each vertex (inner product) of the problem’s graph is a graph with vertices corresponding to the partial products that constitute the inner product represented by the initial vertex. The Hamming distances between the partial products are assigned as costs to the edges of the inner product graphs. However the ordering of the inner products performed in the first step results in the determination of the partial products that will be evaluated first and last for each inner product. Thus the derivation of the minimum cost ordering of the partial products of an inner product can be formulated as a restricted minimal spanning tree problem. Only an open path over the vertices of the inner product graph (partial products) must be found. The problem is restricted since the starting and ending vertices of the path are determined by the inner product ordering. An exact solution can be found if the number of the vertices of the inner product graph is relatively small. Two widely known algorithms for the solution of the minimal spanning trees problem are the algorithms of Kruskal [73] and Prim [73].

In all cases the costs of the edges are bounded in the same way as for the convolutional algorithms. In all cases the costs of the edges are well-bounded positive real numbers. Their lower bound is zero and the upper bound equals to the sum of the number of bits used for the representation of the coefficients and the representation of data terms. If the simpler cost function is used the costs of the edges are well-bounded positive integer numbers. Their lower bound is zero and the upper bound equals to the number of bits used for the representation of the coefficients.

Parallel output computation

The parallel computation of the output points of a transformation of length \( N \times M \) is described by the following equation, for \( j=0,1,\ldots,M-1 \) for all output points \( y_{f(i,j)} \):

\[
y_{f(i,j)} = c_{f(i,j)}, g(j) x_{g(j)} \quad (10)
\]
where y’s are the output points, c’s are the coefficients and x’s are the elements of the input data vector. When the output terms of the basic computation of the transformational algorithms are computed in parallel each input data term is accessed only once from the memory element where it is stored and its contribution to all the output terms is computed. The computation is performed according to the ordering functions f(i,j), g(j). The ordering functions f(i,j), g(j) are one-by-one and onto and in the original case f(i,j)=i, g(j)=j. The f(i,j) ordering functions correspond to the rows of the coefficient matrix (one per column of the coefficient matrix) and the g(j) ordering function corresponds to the columns of the coefficient matrix. The f(i,j) functions determine the order in which the partial products (using a specific input data term) of the output terms are computed for each column. The g(j) function determines the order in which the input data are read from the memory (and the sets of partial products using the same data are computed).

The total hamming distance of the sequence of evaluation of the N×M partial products required for the N output terms as determined by the ordering functions f(i,j), g(j), is given by the following equation:

\[
Total\ HD = \sum_{g(j)=0}^{M-1} \sum_{f(i,j)=0}^{N-1} HD\left(p_{f(i,j),g(j)}\cdot p_{f(i,j)+1,g(j)}\right) + HD\left(p_{N-1,g(j)}\cdot p_{0,g(j)+1}\right)
\]

where \(p_{6(i,j)g(j)}\) is the partial product \(c_{6(i,j)g(j)}\). The term \(HD(p_{N-1,g(j)}\cdot p_{0,g(j)+1})\) denotes the hamming distance between the last partial product of the column \(g(j)\) and the first partial product of the column \(g(j)+1\). This term is included because after the computation of the last partial product \((p_{N-1,g(j)})\) for the output term \(y_{N-1}\) the first partial product \((p_{0,g(j)+1})\) for the term \(y_0\) must be computed. In this case the Hamming distance for a pair of partial products includes only coefficient related (static) information. Data related activity information (AHD between two consecutive data samples) is not included since the data term remain the same for N consecutive partial products (j=constant for i=0,1,…N-1). The aim of the proposed methodology is to derive the ordering functions f(i,j), g(j) that minimize the total hamming distance as described in equation 11. There are no data dependencies between sets of partial products and partial products belonging to the same set. Thus the problem of deriving minimum switching cost ordering functions for the sets of partial products and the partial products of the basic computation is equivalent to deriving minimum switching cost schedules for the sets of partial products and the partial products on the single multiply/accumulator.

The formulation of the computation-reordering problem in this case is similar to that of case 1 formulation. The inner products of sequential output computation case are replaced by sets of partial products using the same input data term. The costs assigned to the edges of the graphs are bounded positive integers and not real numbers. This is because no data related information is included this time. Their lower bound is zero and the upper bound equals to the sum of the number of bits used for the representation of the coefficients.

1.6 Experimental Results for Single Functional Unit Implementations

The proposed techniques have been applied to several digital signal-processing algorithms. Bit-true simulations have been carried out. The results for the different algorithms are described in the following sub-sections. Bit parallel implementations on single multiply-accumulator data paths were assumed. It is assumed that two different busses one for data and one for coefficients connect the corresponding storage elements to the functional units. Two’s complement fixed point arithmetic was assumed for all cases. The power consumption reduction is approximated by the reduction of the switching activity in both the data and coefficient busses. It is assumed that the capacitances of the busses transferring data and coefficients to the functional units are equal (transitions in the two busses are of equal importance). For all cases – denotes savings in switching activities resulted by the application of the proposed transformation, while + denotes penalties introduced by the reordering of computation.

1.6.1 FIR filters

The results of the application of the proposed techniques to several FIR filters are included in Table 1. The first two filters (63 and 14 taps) are band pass filters and were simulated using random data. The coefficients were represented using 16 bits while 12 bits represented data. For the outputs 16 bits were used. Because of the random type of the data, no information related to data was used for the reordering of computation. The effect of the computation reordering on the activity of the data bus is random (small
reduction in the first case – small increase in the second). The remaining filters are low-pass wavelet filters. Real image data were used for simulation. For the representation of the data 8 bits were used while for the coefficients and the outputs 16 bits were used. In this case the effect of the proposed methodology on the data activity is clearer. The computation reordering destroys the data correlation and introduces significant penalty although data related information was used. However the penalty introduced is much smaller than the savings in coefficient activity leading to significant total switching activity and power savings.

<table>
<thead>
<tr>
<th>Filter</th>
<th>Data</th>
<th>Change in coefficient activity (# transitions-%)</th>
<th>Change in data activity (# transitions-%)</th>
<th>% change in total bus activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 tap</td>
<td>Random</td>
<td>-180181 (-62%)</td>
<td>-1827 (-0.4%)</td>
<td>-40%</td>
</tr>
<tr>
<td>14 tap</td>
<td>Random</td>
<td>-115296 (-75%)</td>
<td>+1321 (+0.9%)</td>
<td>-37%</td>
</tr>
<tr>
<td>15 tap</td>
<td>Image</td>
<td>-1127779 (-52%)</td>
<td>+503139 (+35%)</td>
<td>-28%</td>
</tr>
<tr>
<td>11 tap</td>
<td>Image</td>
<td>-745479 (-48%)</td>
<td>+283735 (+25%)</td>
<td>-19%</td>
</tr>
<tr>
<td>9 tap</td>
<td>Image</td>
<td>-496987 (-43%)</td>
<td>+194244 (+21%)</td>
<td>-17%</td>
</tr>
<tr>
<td>8 tap</td>
<td>Image</td>
<td>-152918 (-13%)</td>
<td>+75374 (+15%)</td>
<td>-12%</td>
</tr>
<tr>
<td>6 tap</td>
<td>Image</td>
<td>-133801 (-14%)</td>
<td>+13056 (+2%)</td>
<td>-12%</td>
</tr>
</tbody>
</table>

Table 1: Experimental results for FIR filters

1.6.2 Two dimensional wavelet transforms

The proposed techniques have been applied to the typical three-level two-dimensional wavelet transform. Six different filter sets have been used and simulated. The EPIC filters are described in [74], the Daubechies and Johnston’s filters are described in [75]. The simulation results for two grayscale test images (Lena and Man) are included in Tables 2, 3.

<table>
<thead>
<tr>
<th>Filter set</th>
<th>Change in coefficient activity (# transitions-%)</th>
<th>Change in data activity (# transitions-%)</th>
<th>% change in total input activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPIC – 15 tap</td>
<td>-2030019 (-51.8%)</td>
<td>+2698508 (+12%)</td>
<td>-29</td>
</tr>
<tr>
<td>EPIC – 11 tap</td>
<td>-13418613 (-48.1%)</td>
<td>+954075 (+5.4%)</td>
<td>-28</td>
</tr>
<tr>
<td>EPIC – 9 tap</td>
<td>-8945772 (-43.3%)</td>
<td>+853493 (+6.5%)</td>
<td>-24</td>
</tr>
<tr>
<td>Daubechies – 6 tap</td>
<td>-2408421 (-14%)</td>
<td>+281648 (+3.1%)</td>
<td>-9</td>
</tr>
<tr>
<td>Daubechies – 8 tap</td>
<td>-2752521 (-13.3%)</td>
<td>+646421 (+5.4%)</td>
<td>-9.5</td>
</tr>
<tr>
<td>Johnston’s – 8 tap</td>
<td>-5849097 (-34.3%)</td>
<td>+599234 (+5.1%)</td>
<td>-17</td>
</tr>
</tbody>
</table>

Table 2: Simulation results for the 2-D wavelet transforms-test image Lena

<table>
<thead>
<tr>
<th>Filter set</th>
<th>Change in coefficient activity (# transitions-%)</th>
<th>Change in data activity (# transitions-%)</th>
<th>% change in total input activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPIC – 15 tap</td>
<td>-2030019 (-51.8%)</td>
<td>+2686524 (+11.1%)</td>
<td>-29</td>
</tr>
<tr>
<td>EPIC – 11 tap</td>
<td>-13418613 (-48.1%)</td>
<td>+1037449 (+5.4%)</td>
<td>-27</td>
</tr>
<tr>
<td>EPIC – 9 tap</td>
<td>-8945772 (-43.3%)</td>
<td>+927875 (+6.2%)</td>
<td>-23</td>
</tr>
<tr>
<td>Daubechies – 6 tap</td>
<td>-2408421 (-14%)</td>
<td>+314922 (+3.2%)</td>
<td>-8</td>
</tr>
<tr>
<td>Daubechies – 8 tap</td>
<td>-2752521 (-13.3%)</td>
<td>+659390 (+5%)</td>
<td>-8</td>
</tr>
<tr>
<td>Johnston’s – 8 tap</td>
<td>-5849097 (-34.3%)</td>
<td>+624293 (+4.9%)</td>
<td>-16</td>
</tr>
</tbody>
</table>

Table 3: Simulation results for the 2-D wavelet transforms-test image Man

For the coefficients, a 16-bit representation was used. The input data have been represented by 8 bits while for the representation of the wavelet coefficients 16 bits were used. The switching activity in the coefficient bus is significantly reduced. The computation reordering introduces a penalty (worst case 12%) in the switching activity in the data bus because data correlation is destroyed. Specifically the reordering of computation introduces switching activity penalty in the inputs of the three levels of the wavelet decomposition because at these points the correlation of data is high. In the intermediate and output stages of the decomposition levels data are less correlated and thus the result of the reordering is more random
(introduces either small savings or small penalties in switching activity). However, the penalty in data activity introduced is much smaller in comparison to the savings achieved.

Two basic parameters affecting the result of the computation reordering on the bus switching activity in the case of the wavelet transforms: a) The filter length and b) The symmetry of the filter. From Tables 2, 3 it is obvious that as the length of the filter increases the savings in coefficient activity increase but the penalty in data activity increases also. In cases of symmetrical filters the savings in coefficient activity by the application of computation reordering are higher.

1.6.3 Fourier transform

The proposed techniques have been applied to the one dimensional 8-point Discrete Fourier Transform (DFT) and three common one dimensional Fast Fourier Transforms (FFT) [76], the 9-point PTL FFT, the 7-point Singleton FFT and the 9-point Swift FFT. Real image data were used for the simulation. The bit-width of the input data was 8 while 16 bits were used for the representation of the output fourier data. The simulation results for both sequential and parallel computation cases and for several coefficients bit-widths are shown in Figures 3, 4.

In the case of sequential output computation a penalty in data switching activity is introduced in the DFT (much smaller than the savings achieved in coefficient activity). The FFTs effectively reduce the length of the input data vector through a preprocessing stage to reduce the number of operations required leading to small changes (either savings or penalty) in the data switching activity by the application of the computation reordering. Although the length of the FFTs is relatively small the reordering of computation introduces significant savings in the coefficient bus. For the parallel output computation case large switching activity savings are achieved in the data bus. The savings in the coefficient bus are of the same order as for the sequential output computation case. The savings percentage is not heavily affected by the changes of the coefficient bit-width.

1.6.4 Discrete cosine transform

The proposed techniques have been applied to the 8 and 16 point 1-D DCT. The fast DCT that decomposes an N-point input vector to two N/2 point sequences has been used. Real speech data were used for simulation. The simulation results are shown in Figures 3, 4. The proposed techniques have been also applied to the 2-D DCT computed using the simple row column decomposition and the fast technique described in [75]. For simulation image data were used. In all cases 8 bit input data, 16 bit coefficients and 16 bit output data were assumed. In the row column computation case the application of the proposed techniques reduced the bus activity by 18% while for the case of the fast DCT implementation the switching activity was reduced by 22%.

As the effective length of the input data vector is increased the penalty introduced by the reordering of computation is increased (row-column 8-point DCT). The reduction of the effective length of the input data vector by the preprocessing step of the fast algorithm leads to smaller changes (either savings or penalties) of the switching in the data bus. The results for the 1-D DCTs prove that the savings in bus switching activity do not always increase as the transform length increases. The same is true for the coefficient bit-width. Increase of the coefficient bit-width does not always lead to higher savings percentage. Changes of the coefficient bit-width lead to small changes of the savings percentage in the total bus activity. For the 2-D DCT the behavior of the outputs of the rows DCTs (row-column decomposition is always assumed for DCT computation even when a fast algorithm is used) is random since these data are less correlated. This leads to smaller penalties introduced by the computation reordering in the data bus during computation of the columns DCTs.
1.6.5 Comparison of cost functions

As already mentioned in section 4 there two possible cost functions that can be used to drive the proposed scheduling techniques. The first cost function includes both data and coefficient related information. Simulations with real application data are used to collect the data related information. This introduces a significant penalty as far as the speed of the scheduling procedure is concerned. The second cost function includes only the static coefficient-related information eliminating the need for simulations and increasing the speed of the scheduling procedure. To explore the scheduling procedure speed - quality of the produced schedule trade-off both cost functions have been used to schedule some of the kernels described in the previous section. In all cases the coefficient bit-width was supposed to be 16 bits. The comparison of the produced schedules in terms of bus switching activity reduction (schedule quality) is illustrated in figure 5.
From the results it is clear that the use of both data and coefficient information during scheduling leads to better results in terms of bus switching activity. However the differences in terms of total savings percentage is small especially for the cases of kernels of small length such as the FFTs. Differences in percentage savings tend to increase as the length of the kernels increase. In such cases penalties in data bus switching activity are introduced because of the decrease of the data correlation. This decrease of data correlation is larger when only the static information is used during scheduling. The following heuristic rules can be used to decide on which cost function to be used to drive the scheduling.

a. The cost function including only the static component may be used to trade-off minor reductions of the schedule quality for significant increase of the scheduling procedure in the following cases:
   i) The length of the kernel is relatively small.
   ii) The application data have random nature.
   iii) The capacitance of the coefficient busses is known/expected to be larger than that of the data busses.

a. The cost function including both the static and the dynamic component may be used to derive a schedule of better quality in the following cases:
   i) The length of the kernel is relatively large.
   ii) The application data are strongly correlated.
   iii) The capacitance of the data busses is known/expected to be larger than that of the coefficient busses.

1.7 Proposed Synthesis Techniques for Multiple Functional Units Implementations

The synthesis techniques described in section 5 assume that each basic computation is realized on a single functional unit (multiply/accumulator). Under this assumption, the result of the proposed techniques is independent from the number of functional units used for the implementation of an algorithm. This is true for example when several functional units are present and used to compute different basic computations in parallel. The synthesis techniques that will be presented in the next sub-sections handle the cases, where basic computations are realized on more than one functional units.

1.7.1 Multiple functional unit architectures

When multiple functional units are present, two main options exist for storage and interconnection organization and they are described in figures 6 and 7. In the first case, data and coefficients are stored in central storage elements. These storage elements are multi-port and connected through multiple buses (or a
single wide bus) to the functional units, to allow parallel computation. The data memory contains at least the data required by a basic computation. The second architecture is more local. Each functional unit has local data and coefficient memories. These memories are single-port and connected through a single local bus to the corresponding functional unit. Both data and coefficient memories may contain information required only for a part of a basic computation. A distributed memory architecture is in general more power-efficient than a centralized one [1]. The proposed synthesis technique can be applied in both cases.

**Fig.6: Centralized memory and bus organization**

**Fig.7: Localized memory and bus organization**

Two options exist as far as the configuration of multiple functional units is concerned. These configurations are shown in figures 8 and 9.

**Fig.8: MAC-based configuration**
Multiplier 1 \hspace{1cm} Multiplier 2 \hspace{4cm} Multiplier K
\hspace{2cm} \downarrow \hspace{2cm} \downarrow \hspace{2cm} \downarrow
\hspace{4cm} \text{Accumulator}

Fig. 9: Multiplier-based configuration

For the convolutional algorithms there are no critical differences between realizations in the two different configurations. However, realization using the first configuration would require an extra global accumulator. For the transformational algorithms the first configuration allows parallel evaluation of the required inner products. (When an inner product is computed using more than one multiply/accumulators, an extra accumulator must be introduced). Using the second configuration the inner products of the basic computation must be computed sequentially. The second configuration is less flexible, generates dependencies, and reduces the freedom as far as scheduling and assignment of partial products is concerned. The first configuration will be referred to as MAC-based configuration, while the second one as multiplier-based.

1.7.2 Convolutional algorithms
In the general case, a basic convolution computation of this category can be easily implemented in parallel on K functional units in the following way:

$$Y_n = \sum_{i=0}^{a_1-1} C_i \times X_{n-i} + \sum_{i=a_1}^{a_2-1} C_i \times X_{n-i} + \ldots + \sum_{i=a_{k-1}}^{N-1} C_i \times X_{n-i} \quad (12)$$

In most cases this is not a good solution in terms of input switching activity. The proposed techniques consist of the following steps:

a) **Assignment:** Each of the N partial products is assigned to one of the K functional units available. The only restriction is the load balancing among the functional units. Since no data dependencies exist between partial products, this step is independent from the order of evaluation of the partial products. The N partial products are clustered to K groups of highly correlated partial products. This will lead to switching activity reduction. The problem of the optimal clustering is intractable. Two heuristics are proposed:

**Heuristic 1:** Schedule the basic computation assuming realization on a single functional unit as described in section 5 and then perform a clustering sequentially, in the same way as shown in eqn. 12.

**Heuristic 2:** Order the N partial products with respect to the values of their respective coefficients and cluster the resulting sequence into K sub-sequences.

More sophisticated algorithms can be used for clustering the partial products into small switching activity groups. The above heuristics were presented to demonstrate the feasibility of the proposed technique. They also achieve a good trade-off between clustering time and switching activity reduction, as it will be shown in the following section.

b) **Scheduling:** In this step, a schedule is derived for the partial products assigned to each functional unit. This is achieved by deriving a scheduling function $f$ for each functional unit that minimizes the following expression:

$$\sum_{f(k)=0}^{N-1} \text{HD}(p_{f(k)}, p_{f(k)+1}) + \text{HD}(p_0, p_{N-1}) \quad (13)$$
Where $N_1$ is the number of partial products assigned to each functional unit. This normally should be equal to $\lceil N/M \rceil$ or $\lceil N/M \rceil - 1$. The problem of derivation of $f$ is formulated as a Travelling Salesman Problem (TSP) as described in section 5.

### 1.7.3 Transformational algorithms

The configuration of the functional units is critical in this case, since it affects the freedom as far as the ordering of computation is concerned.

**MAC-based configuration**

It is assumed that no splitting of the basic computation’s inner products to different functional units (MACs) is performed and that load balancing is preserved. Implementation of a basic computation in $K$ functional units creates $K$ sub-basic computations of the form:

$$
\begin{bmatrix}
Y_i \\
Y_{i+N/K-1}
\end{bmatrix} = \begin{bmatrix}
C_{(i)} & C_{(i+1)} & \ldots & C_{(M-1)} \\
C_{(i+N/K)} & C_{(i+N/K+1)} & \ldots & C_{(i+N/K+M-1)}
\end{bmatrix} \times \begin{bmatrix}
X_0 \\
X_{M-1}
\end{bmatrix}
$$

(14)

where $i = j \times (N/K)$, $j = 0, \ldots, K - 1$. Each of these sub-basic computations can be assigned to one functional unit. This is again not the best solution.

Two main cases exist:

a) $K = N$. The number of available functional units is equal to the number of inner products. The assignment is straightforward one inner product is assigned to each functional unit. The scheduling of the partial products in each functional unit is performed in the same way as for the convolutional algorithms.

b) $K < N$. More than one inner products must be assigned to each functional unit.

**Assignment:** In this step groups of correlated inner products are created to reduce the switching activity, when passing from one inner product computation to another. Finding the minimal transition activity grouping of the inner products is too complex. A heuristic technique is adopted for this reason. In section 5 a technique for minimal activity ordering of the inner products of the basic computation was described. This technique finds the minimum activity transition for all possible pairs of inner products (i.e minimal switching activity pair of partial products belonging to each pair of inner products). The minimal transition activity ordering of the inner products is determined by solving the TSP problem on a graph with vertices corresponding to the inner products and edges weighted by the values of minimal cost transitions. This technique targeted realizations of the basic computation on the same functional unit. In the case of multiple functional units, the previously described ordering is performed in a first step. The resulting sequence of inner products is then divided sequentially to $K$ groups of $N/K$ inner products. Each of the groups is assigned to each functional unit.

**Scheduling:** It is assumed that the inner products assigned to each functional unit are computed sequentially. Thus the assignment step produces a schedule for the inner products as well. The task of the second step is to schedule the partial products within each inner product assigned to the same functional unit. This is achieved by using the same approach as for the algorithms of the first category. For this case, the problem of minimal activity ordering of the partial products is restricted by the output of the assignment step. The problem can be formulated and solved as a minimal spanning tree problem.

**Multiplier-based configuration**

In this case, the inner products are computed sequentially. It will be assumed that the size $M$ of the inner products is equal to the number of the available functional units (multipliers). If this is not the case, each inner product can be divided to more than one parts and the same strategy can be followed. The following heuristic strategy is used for the assignment and scheduling of the partial products on the multipliers. For all the pairs of inner products the hamming distances between all possible pairs of partial products (one partial product from each inner product) are computed. The minimum cost connection for every pair of inner products is determined. The connection cost between two inner products is defined as the sum of hamming distances of the pairs of partial products assigned to the same multiplier. Having determined the
minimum connection cost for all the possible pairs of inner products, the problem of finding the optimal scheduling in terms of switching activity at the multipliers inputs is formulated as a Travelling Salesman Problem on a graph with the inner products as vertices. The costs of the graph’s edges are the hamming distances of the minimum cost connections of the inner products corresponding to the vertices.

1.8 Experimental Results for Multiple Functional Units Implementations

The effect of the proposed synthesis technique was evaluated by applying it on some typical DSP algorithms. Data paths with 2, 3, and 4 functional units were assumed. For the first category of algorithms both assignment heuristics have been used. The three FFT transforms are typical building blocks of larger FFTs. For the DCTs the fast implementation described in [75] was used. The Singleton FFT has been only mapped to three functional units while for the rest transformational algorithms mappings to two and four functional units were performed. For all algorithms image data have been used for simulation. The bit width of the coefficients was assumed to be 12 bits. For the functional units, bit parallel implementation was assumed. The results of the proposed synthesis technique in terms of the percentage reduction of the switching activity at the inputs of the functional units are presented in tables 4, 5, and 6.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>2 FUs</th>
<th>3 FUs</th>
<th>4 FUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 tap Fir filter</td>
<td>47.3%</td>
<td>29.3%</td>
<td>42.3%</td>
</tr>
<tr>
<td>11 tap Fir filter</td>
<td>26.1%</td>
<td>18%</td>
<td>44.2%</td>
</tr>
<tr>
<td>9 tap Fir filter</td>
<td>42.2%</td>
<td>24.1%</td>
<td>46.6%</td>
</tr>
<tr>
<td>8 tap Fir filter</td>
<td>30.2%</td>
<td>22.2%</td>
<td>16.4%</td>
</tr>
<tr>
<td>6 tap Fir filter</td>
<td>21.5%</td>
<td>19.4%</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4: Switching activity savings (%) at the inputs of functional units for the first category of algorithms using the first assignment heuristic

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>2 FUs</th>
<th>3 FUs</th>
<th>4 FUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>15 tap Fir filter</td>
<td>49.1%</td>
<td>31.6%</td>
<td>46.2%</td>
</tr>
<tr>
<td>11 tap Fir filter</td>
<td>39.7%</td>
<td>32%</td>
<td>31.4%</td>
</tr>
<tr>
<td>9 tap Fir filter</td>
<td>39.9%</td>
<td>24.5%</td>
<td>59%</td>
</tr>
<tr>
<td>8 tap Fir filter</td>
<td>31.9%</td>
<td>23.1%</td>
<td>17.9%</td>
</tr>
<tr>
<td>6 tap Fir filter</td>
<td>21.1%</td>
<td>18.1%</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5: Switching activity savings (%) at the inputs of functional units for the first category of algorithms using the second assignment heuristic

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>2 FUs</th>
<th>3 FUs</th>
<th>4 FUs</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-pt. Singleton FFT</td>
<td>-</td>
<td>9.8%</td>
<td>-</td>
</tr>
<tr>
<td>9-pt. Swift FFT</td>
<td>14.1%</td>
<td>-</td>
<td>11.1%</td>
</tr>
<tr>
<td>9-pt. PTL FFT</td>
<td>15%</td>
<td>-</td>
<td>11.2%</td>
</tr>
<tr>
<td>1-D 8-pt. DCT</td>
<td>24.2%</td>
<td>-</td>
<td>15.6%</td>
</tr>
<tr>
<td>2-D 8-pt. DCT</td>
<td>22.6%</td>
<td>-</td>
<td>14.4%</td>
</tr>
<tr>
<td>1-D 16-pt. DCT</td>
<td>16.7%</td>
<td>-</td>
<td>15.2%</td>
</tr>
</tbody>
</table>

Table 6: Switching activity savings (%) at the inputs of functional units for the second category of algorithms

The proposed synthesis technique reduce significantly the switching activity at the inputs of the functional units, although the sizes of the basic computations in the selected algorithms are relatively small, restricting thus the optimization space of the proposed techniques. The results prove that the switching activity savings are not always proportional either to the size of the basic computation or to the number of functional units (this is still true if the results of the single functional unit realizations presented in section 5 are taken into consideration). In all cases the proposed techniques achieve large reductions of the switching activity at the coefficient inputs of the functional units, while in most cases it introduces small penalties in the switching activity of the data inputs of the functional units. As far as the two assignment heuristics for the convolutional algorithms are concerned, the results prove that the second heuristic performs slightly better in more cases, although it uses only coefficient information. This proves that the static information may affect more the output of the proposed techniques than the simulation-based
information on data. The proposed technique was also applied to a 2-D fast DCT realized on a multiplier-based configuration of the data path with 4 multipliers. The input switching activity was reduced by 25% in this case.

1.9 Evaluation of the Effects of the Proposed Synthesis Techniques

The proposed synthesis techniques based on the reordering of computation, reduce the total switching activity (and power consumption) in the busses connecting the data and coefficient storage elements to the functional units. This leads to significant system power consumption reduction since the capacitance of the busses is very large in comparison to that of the functional units. This is especially true for deep submicron technologies. The switching activity in the coefficient memory periphery unit is also reduced.

The proposed techniques effectively reduce the switching activity at the inputs of the computational units. The switching activity of a circuit depends on the internal node structure and on the switching at the inputs. At the high levels of abstraction the detailed internal architecture of the computational units is not known and the units are treated as black boxes. The switching activity inside the computational units and thus the power consumption is proportional to the input switching activity [70]. In general at the high levels of the design abstraction the power consumption is estimated based on the total input switching activities [70]. The reduction of switching activity inside the computational units is directly proportional to power consumption reduction since the other factors affecting power consumption (supply voltage, frequency of operation, and capacitive load) do not change by the application of the proposed methodology. Thus it can be expected that the proposed techniques lead to important power reduction inside the computational units as well.

The change in the order in which data and coefficients are transferred to the functional units (as compared to the original case) does not introduce any addressing penalty in the target custom hardware realizations. Data are transferred to the registers in the same order as in the original case. The registers are accessed by signals generated directly by the control unit and thus no address generation penalty is introduced. As far as the activity of the address lines is concerned, activity reduction techniques can be used as in the original case. In the future techniques for register address derivation and assignment to the available/required registers taking into consideration the new schedules (order of register accesses) in order to reduce the address lines activity will be explored. Furthermore the inclusion of the address lines switching activity in the cost function driving the scheduling procedure will be investigated. The same comments apply for the case of coefficient register storage as well. In cases where coefficients are stored in addressable memories (not directly addressed by the control unit-require address generation hardware) they can be arranged in the memory according to the new schedule (order of access) to eliminate address generation and address lines switching activity penalties in comparison to the original schedule.

The application of computation reordering may lead to address generation overhead in programmable architectures. Strategies for simplification of the address generation in typical digital signal processing applications (e.g. FIR filters) like modulo addressing, are present in many programmable DSP processors. These techniques may not work well when computation reordering takes place leading to an increase of the instruction cycle budget because of address generation overhead. The execution of extra instructions will offset the savings achieved by the application of the computation reordering. To explore the effect of the application of the reordering on programmable architectures some typical DSP kernels were mapped on ARM (general purpose processor) and on Phillips TriMedia (multimedia processor) before and after the application of the computation reordering. The results are summarized in tables 7, 8:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>#Executed Basic Computations</th>
<th>Initial - #cycles</th>
<th>Reordered - #cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fir filter (9-point)</td>
<td>65536</td>
<td>161920803</td>
<td>159493584</td>
</tr>
<tr>
<td>Fir filter (8-point)</td>
<td>65536</td>
<td>143356662</td>
<td>143320965</td>
</tr>
<tr>
<td>Singleton FFT (7-point)</td>
<td>9216</td>
<td>58472006</td>
<td>58598715</td>
</tr>
<tr>
<td>Swift FFT (9-point)</td>
<td>7168</td>
<td>74168707</td>
<td>74139843</td>
</tr>
<tr>
<td>Fast 2-D DCT (8-point)</td>
<td>16384</td>
<td>180721803</td>
<td>180326918</td>
</tr>
<tr>
<td>Fast 1-D DCT (8-point)</td>
<td>8192</td>
<td>90201011</td>
<td>89833538</td>
</tr>
</tbody>
</table>

Table 7: Results of mapping DSP kernels on ARM
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>#Executed Basic Computations</th>
<th>Initial - #cycles</th>
<th>Reordered - #cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fir filter (9-point)</td>
<td>65536</td>
<td>7538237</td>
<td>7538245</td>
</tr>
<tr>
<td>Fir filter (8-point)</td>
<td>65536</td>
<td>7145152</td>
<td>7145160</td>
</tr>
<tr>
<td>Singleton FFT (7-point)</td>
<td>9216</td>
<td>6896597</td>
<td>6896577</td>
</tr>
<tr>
<td>Swift FFT (9-point)</td>
<td>7168</td>
<td>7575071</td>
<td>7575079</td>
</tr>
<tr>
<td>Fast 2-D DCT (8-point)</td>
<td>16384</td>
<td>15043055</td>
<td>14993917</td>
</tr>
<tr>
<td>Fast 1-D DCT (8-point)</td>
<td>8192</td>
<td>9423572</td>
<td>9399007</td>
</tr>
</tbody>
</table>

**Table 8: Results of mapping DSP kernels on TriMedia**

The sizes of the algorithms codes before and after the application of the reordering are the same. The only differences are in the order of the sum-of-product computation. Loop unrolling have been performed in both cases to ensure register storage of the data terms required for the basic computations. An important observation is that the application of the proposed scheduling techniques does not always leads to increase of the number of cycles for both processors. In some cases the total number of cycles is reduced.

Specifically for ARM, in five out of the six cases the application of the computation reordering techniques improved performance in terms of cycles. The only case where a penalty is introduced is the Singleton FFT (0.002%). The larger improvement is obtained in the 9 point FIR filter (0.015%).

As far as TriMedia is concerned the differences (3 increases-3 decreases) in number of cycles between the initial and reordered cases are very small (maximum 0.3% decrease after the application of the reordering for the 2-D DCT and minimum 0.001% increase after the application of the reordering for the FIR filters and the Swift FFT). The surprising result is that the largest changes in number of cycles (1-D and 2-D DCTs) are reductions after the application of the computation reordering.

The above results prove that the application of the proposed scheduling techniques based on computation reordering does not always lead to increase of the number of execution cycles (because of addressing overhead) in programmable architectures. When an overhead is introduced this is very small while in some cases even reduction of the number of cycles is achieved.

As far as the registers address lines activity is concerned further investigation is required. Specifically in programmable architectures the register addresses are fixed. Techniques for assigning the data terms (and even the coefficients) of basic computations to the registers of a programmable processor in order to minimize the switching activity of the register address lines should be developed. Such techniques are still useful in cases where no computation reordering is performed. Inclusion of the register address lines switching activity in the cost function driving the reordering will also be explored.

### 1.10 Comparison with other Techniques for Low Power DSP

Two techniques for power optimization of sum-of-product computation are described in [38, 49]. The technique described in [38] derives an order for the evaluation of partial products in sum-of-product computation. However it targets dedicated implementations of linear DSP computation (one functional unit/operation) and thus it is less general than the techniques described in the previous sections. The approach described in [49] simply proposes the parallel output computation for the transformational type of algorithms to reduce the number of memory accesses in a programmable processor context. No further optimization with respect to the order of computation is proposed. Thus this approach is significantly limited in comparison to the proposed techniques. The scheduling and assignment techniques described in [12-16, 18, 30] are based on simulation and they do not exploit the fact that for sum-of-product computation one of the two inputs of the partial products evaluation operations is statically known, to get results of better quality in terms of power. The proposed scheduling and assignment techniques can be combined with the variable voltage scheduling techniques [7-10] and the power management technique proposed in [11]. The transformation based approaches described in [2, 3] and the interconnect power optimization techniques proposed in [19-22] can be used as a first step before the application of the proposed synthesis techniques. The synthesis techniques described in [28] use Hamming distance as criterion but they target reduction of switching activity in the functional units. Furthermore no formulation for design automation is proposed. The algorithm selection approach described in [41] as well as the transformational based approaches described in [39, 40, 44, 45] can be used as a preprocessing step before the application of the computation reordering synthesis techniques described in the previous sections. The same is true for the filtering optimization approaches described in [50-54].
2. Techniques for Switching Activity Estimation in FIR Filters Realizations

2.1 Introduction

Power dissipation has become a major concept in VLSI design at last years. The need for low power architectures imposes the development of CAD tools which will incorporate efficiently power estimators. It is well known that the dominant factor of power dissipation in CMOS circuits is the dynamic power dissipated during charging-discharging the parasitic capacitance. Since dynamic power dissipation depends on the number of transitions occurring at the capacitive nodes, calculation of the bit-level signal transitions is a key requirement in most of the power estimation techniques.

In this work, a novel method for the accurate calculation of the transition activity at the nodes of a MAC architecture which implements FIR filters, is introduced. The results of this method can be used as inputs to other methods, suitable for the estimation of power dissipated into the arithmetic operators, in order the overall power needed for the filter implementation to be calculated. The proposed method follows the approach described in [78] and the transition activity at each node is derived from the statistics of the signal values at that node. Consequently, the knowledge of the signal statistics at the nodes of the MAC architecture is required and an efficient technique is proposed in order to extract these values. This technique is based on the fact that the signal sequences which appear at each node of the MAC result by a multiplexing in time of signal sequences with already known statistics. This multiplexing mechanism is mathematically formulated and the corresponding signal statistics are analytically determined.

The proposed method is derived for input signals which can be described by Gaussian processes. The DBT model is adopted for the bit-level transition activity of a signal word and the breakpoints, as they are modified by the multiplexing mechanism, are derived by the signal statistics. Using the word-level correlation coefficient and the breakpoints, the transition activity at each bit-line of the architecture is calculated.

2.2 Preliminaries

Let \( x(n) \) be a real input signal of a Digital Signal Processing (DSP) system. In general this signal can be modeled by a random Gaussian process with mean value \( \mu \), variance \( \sigma^2 \), and correlation factor \( \rho \). The lag-\( i \) temporal correlation \( \rho(i) \) of \( x(n) \) is defined as:

\[
\rho(i) = \frac{E[x(n) x(n-i)] - \mu^2}{\sigma^2} \quad (15)
\]

Here, we will be interested mainly in \( \rho(1) \) and therefore it will be denoted via the simplified notation \( \rho \).

Let now \( b_{ci}(n) \) be the \( i \)th bit of the word-level signal \( x(n) \). With \( p_i^1 \) and \( p_i^0 \) we define the probability of \( b_{ci}(n) \) to be 1 or 0, respectively, while with \( \rho_{ci} \) we define the bit-level correlation factor:

\[
\rho_{ci} = \frac{E[b_{ci}(n) b_{ci}(n-1)] - (p_i^1)^2}{p_i^1(1-p_i^1)} \quad (16)
\]

The transition activity of a bit can be expressed as a function of the bit-level probabilities and the correlation factor:

\[
t_i = 2p_i^1(1-\rho_{ci}) \quad (17)
\]

The word-level transition activity, \( T \), can be used as a measure of the overall activity at an architectural node (where \( B \) is the number of bits of the node):

\[
T = \sum_{i=0}^{B-1} t_i \quad (18)
\]

The bit-level correlation factor is accurately modeled by the Dual Bit Type (DBT) model [79] where a word is divided in three regions of continuous bits referred to as the LSB, linear and MSB regions. The breakpoints \( B_{L0} \) and \( B_{L1} \) separate the LSB from the linear region and the linear from the MSB region, respectively, and are given by:
\[ \text{BP}_0 = \log_2 \sigma + \log_2 \left[ \sqrt{1 - \rho^2} + \frac{|\rho|}{8} \right] \]
\[ \text{BP}_1 = \log_2 (|\mu| + 3\sigma) \]

Since the form of the transition activity per bit position follows that of the correlation coefficient with the same breakpoints, a corresponding model for the transition activity can be derived as:

\[ t_i = \begin{cases} 
2p_i p_0 & \text{if } i < \text{BP}_0 \\
\frac{(t_{mb} - 2p_1 p_0) + 2p_1 p_0 \text{BP}_1 - t_{mb} \text{BP}_0}{\text{BP}_1 - \text{BP}_0} & \text{if } \text{BP}_0 \leq i < \text{BP}_1 \\
& \text{if } i \geq \text{BP}_1 
\end{cases} \]

where \( t_{mb} \) is the activity of the MSB (the sign bit in a two’s complement representation). Recently in [80], \( t_{mb} \) has been accurately expressed, for zero-mean Gaussian signals, as a function of the word-level correlation factor. According to that:

\[ t_{mb} = \frac{1}{\pi} \cos^{-1}(\rho) \]

From the above analysis it is obvious that in order to estimate the activity and thus the power consumed during the data processing in a DSP architecture the signal statistics at any node of the architecture have to be initially determined.

### 2.3 Propagation of the signal statistics in FIR structures

The method proposed in [78] is followed here for the estimation of the signal statistics at each node of an FIR structure. According to that method, signal statistics are propagated through the basic operators met in these architectures (adder, multiplier, delay element) as follows:

For an adder with two input signals \( x_i(n) \) (\( i=1,2 \)) with statistics \( \mu_i, \sigma_i, \) and \( \rho_i \), the statistics for the output signal \( y_s(n) \) are given as:

\[ \mu_s = \mu_1 + \mu_2 \]
\[ \sigma_s^2 = \sigma_1^2 + \sigma_2^2 + 2E[x_1(n)x_2(n)] - 2\mu_1\mu_2 \]
\[ \rho_s = \frac{\rho_1\sigma_1^2 + \rho_2\sigma_2^2 + E[x_1(n)x_2(n-1)] + E[x_1(n)x_2(n-1)] - 2\mu_1\mu_2}{\sigma_s^2} \]

For a multiplier which has as inputs the signal \( x(n) \), and a constant coefficient \( c \), the statistics for the output signal \( y_p(n) \) are given by:

\[ \mu_p = C\mu \]
\[ \sigma_p^2 = C\sigma \]
\[ \rho_p = \rho \]

The statistics at the output of a delay element are identical to that of its input. Consequently, the way of the propagation of the signal statistics through an FIR structure shown in figure 10 can be easily determined.

The two inputs of the adder of the \( l \)th tap have the form \( x_1 = \sum_{i=0}^{l-1} c_i x(n-i) \) and \( x_2 = c_l x(n-l) \) and thus the signal statistics at its output are given by the following expressions [80]:

\[ \mu_{l-1} = \mu \sum_{i=0}^{l-1} c_i \]
\[ \sigma_{l-1}^2 = \sigma^2 \left( \sum_{i=0}^{l-1} c_i^2 + 2\sum_{i=0}^{l-1} \sum_{j=i+1}^{l-1} \rho (j-i)c_i c_j \right) \]
The values of lag-2, lag-3 .... lag-(l-1) correlation coefficients of the input signal are also required. For simplicity the approximation of lag-j correlation coefficient by $r_j(1)$, which is valid for the most real-life signals, can be used without significant degradation of the accuracy of the method. In this way, the signal statistics at the output of each tap of the filter structure can be estimated and consequently the transition activity at each node of an FIR structure can also be calculated. However, this calculation can not be used for the estimation of the power dissipated during the filter realization because in most cases it doesn’t correspond to a real implementation scheme. Generally, FIR filters are implemented on a single MAC unit and thus the transition activity at the nodes of a MAC unit when it realizes FIR filters has to be determined.

![Fig. 10: FIR filter structure](image)

**2.4 Mapping the FIR filter structure on a MAC**

FIR filters are usually implemented on architectures which are composed by a multiplier and an accumulator, as shown in figure 11. Registers are inserted after the outputs of these data-path operators in order to increase the throughput of the architecture. It must be noted that the activity at the output of the registers (these nodes are of interest here) corresponds to the activity at the output of the arithmetic operators if zero-delay gate model is considered, ignoring the increased activity due to the glitching effect.
Comparing the FIR structure with the way the filter is implemented on the MAC, one can observe that the MAC architecture results after the projection of the filter structure along the direction of the filter order. This means that for a \( k \)-tap FIR filter, \( k \) MAC operations (\( k \) multiplications and additions) are required for every input sample. Consequently, the signal values which appear during an input signal sampling period at the nodes of the filter structure (in the space domain) will appear as a sequence in the time domain at each of the corresponding nodes (resulted after projection) of the MAC unit. For example, during the \( n \)th input signal sampling period the sequence \([x(n), x(n-1), x(n-2) \ldots x(n-k+1)]\) appears at the input of the MAC and during the next sampling period the sequence \([x(n+1), x(n), x(n-1) \ldots x(n-k+2)]\) appears. In order to find the transition activity in the MAC the statistics of the signals which appear at its nodes, after the above multiplexing in time, have to be calculated.

It can be considered that the sequence of the signal values which appears at a node of the MAC during an input signal sampling period, corresponds to the output of a multiplexer which has as inputs the corresponding nodes of the filter structure and takes as control signal a gradually increased number so that to pass to the output the signal values with the proper order. It has also to be mentioned that all these signal values follow normal distributions. Having the statistics of the input signals of the multiplexer, a mathematical approach is followed in order to find the signal statistics at the output of the multiplexer, which corresponds to the signal statistics at a node of the MAC.

The described multiplexing mechanism is a special case of the mixing method referred in [81], and the density function of the multiplexer output sequence \( x \) results as the average value of the inputs densities:

\[
f(x) = \frac{1}{m} \sum_{i=1}^{m} f_i(x) \quad (25)
\]

where \( f_i(x) \) is the density function of the \( i^{th} \) input of the multiplexer. In figure 12 a comparison between the distribution of the actual output values of a multiplexer, which has as inputs twenty four different zero-mean Gaussian processes and that one derived according to eq. (11), is presented. The expected value, \( \mu_x \), and the variance, \( \sigma_x^2 \), of the resulted sequence at the output of the multiplexer are given by:

\[
\mu_x = \frac{1}{m} \sum_{i=1}^{m} \mu_i \quad (26)
\]
\[ \sigma_x^2 = \frac{1}{m} \sum_{i=1}^{m} [\sigma_i^2 + \mu_i^2] - \mu_x^2 \]  

(27)

where \( \mu_i \) and \( \sigma_i^2 \) are the mean value and the variance of the input sequence \( x_i \), respectively. The correlation coefficient of the output sequence is given by:

\[ \rho_x = \frac{E[x(n)x(n-1)] - \mu_x^2}{\sigma_x^2} \]  

(28)

where

\[ E[x(n)x(n-1)] = \frac{\sum_{i=1}^{m} E[x_i(n)x_{i-1}(n)] + E[x_1(n+1)x_m(n)]}{m} \]

Fig. 12: Actual and theoretic probability density functions at the output of the multiplexer

Finally, as it will be shown in the next section, the correlation coefficient at the output of the multiplexer results as a function of the filter’s input signal correlation coefficient.

Another significant effect of this multiplexing is the distortion on the signal values distribution that it provokes. For a multiplexer which has as inputs signal sequences with normal distribution but with different statistics, the resulted output sequence differs significantly from to be a normal distribution, especially in the case that the input sequences have different mean values. Only in the case where the inputs present normal distribution with the same signal statistics (same mean value and variance) the output signal is a process with normal distribution and with the same statistics. This can be easily verified by eq. (11). Indeed, the distribution of the output signal values presented in figure 12, which have been resulted by multiplexing of Gaussian processes with the same mean value (\( \mu = 0 \)) differs from to be a normal distribution. More specifically it narrows around the mean value while keeps its symmetry. This distortion in the distribution shifts the value of the breakpoint \( BP_0 \) towards to the LSB because the effective deviation which now determines the smallest significant range of signal values decreases. Exploiting its symmetry around the mean value a method is presented for estimating the effective deviation which will be used for the calculation of \( BP_0 \).

The only case to have signal sequences with a normal distribution and the same mean value at all the nodes of the filter structure which corresponds to the inputs of the multiplexer, is when a signal sequence with zero-mean normal distribution is applied as input to the filter structure. Consequently, the proposed method has been developed for inputs which are zero-mean Gaussian signals. However, this limitation does not reduce the worth of the method as these signals corresponds to the most real-life signals.
For a signal sequence whose values present a zero-mean normal distribution with a density function \( g(x) \), the breakpoint \( \text{BP}_0 \) is defined as \( \text{BP}_0 = \log s \) (assuming \( r = 0 \)) and determines the number of bits needed for the representation of signal values with the highest probability to appear, i.e., values which are included in the set \( [-\sigma, \sigma] \) (for \( \mu = 0 \)). For all these values it is \( g(x) \geq g(0) = e^{-1/2} \) where \( g(0) \) is the maximum value of the density function. Consequently, the effective deviation, \( \sigma_{\text{eff}} \), of the density function at the output of the multiplexer, \( f(x) \), could be defined as the signal value for which it is \( f(\sigma_{\text{eff}}) = f(0) e^{-1/2} \) where \( f(0) \) can be calculated from eq (11). The resulting effective deviation is applying in eq. (5) and the \( \text{BP}_{0,x} \) for the signal sequence at the output of the multiplexer is calculated.

For the \( \text{BP}_1 \), since the most common signal values at the output of the multiplexer remain within the range determined by the value of the real deviation (\( \text{Rx} = [-3\sigma, 3\sigma] \)), the way of the calculation of the \( \text{BP}_{1,x} \) remains unaffected. Having the values of \( \text{BP}_{0,x} \) and \( \text{BP}_{1,x} \) the transition activity at the output of the multiplexer can be estimated from eq.(6). Results which verify the correctness of the derived formulas for the signal statistics and the breakpoints at the output of the multiplexer are given in the following sections.

### 2.5 Signal statistics at the nodes of the MAC

Let assume that the MAC unit, shown in figure 11, is used for the implementation of a \( k \)-tap FIR filter (figure 10) which filters an input signal \( x(n) \), described by a Gaussian process with signal statistics \( \mu, \sigma^2 \) and \( \rho \). According to the multiplexing mechanism defined in the previous section, the statistics of the signal \( y_1(n) \) at node #1 are given by:

\[
\mu_1 = \frac{1}{k} \sum_{i=0}^{k} \mu_i = \mu
\]

\[
\sigma_1^2 = \frac{1}{k} \sum_{i=1}^{k} (\sigma^2 + \mu^2) - \mu^2 = \sigma^2
\]

\[
\rho_1 = \frac{\text{E}[y_1(n)y_1(n-1)] - \mu_1^2}{\sigma_1^2},
\]

where

\[
\text{E}[y_1(n)y_1(n-1)] = \frac{\sum_{i=2}^{k} \text{E}[x(n-i+1)x(n-i+2)] + \text{E}[x(n+1)x(n-k+1)]}{k}
\]

Since it is:

\[
\text{E}[x(n-i+1)x(n-i+2)] = \rho \sigma^2 + \mu^2
\]

\[
\text{E}[x(n+1)x(n-k+1)] = \rho^k \sigma^2 + \mu^2
\]

the correlation coefficient of \( y_1(n) \) is expressed as:

\[
\rho_1 = \frac{(k-1)\rho + \rho^k}{k}
\]

The simplification \( \rho(i) = \rho^i \), which is valid for the most real-life signals, has been used. Since \( \rho < 1 \), for high filter order the correlation coefficient at node #1 approximates the correlation of the input signal, as it is expected.

At the output of the multiplier, \( y_3(n) \)

Since the signal values and the filter coefficients are mutually independent, it is:

\[
\mu_3 = \text{E}[c_i y_1(n)] = \mu_i \mu_1
\]

where \( \mu_c = \frac{1}{k} \sum_{i=0}^{k-1} c_i \) the average value of the coefficients (at node #2). The variance of \( y_3(n) \) is:
\[
\sigma_3^2 = (\sigma_c^2 + \mu_c^2)(\sigma_1^2 + \mu_1^2) - \mu_3^2 \tag{32}
\]
where \( \sigma_c^2 = \frac{1}{k} \sum_{i=0}^{k-1} c_i^2 - \mu_c^2 \) is the variance of the sequence of the filter coefficients. The correlation coefficient of \( y_3(n) \) is given by:

\[
\rho_3 = \frac{(\rho_c \sigma_c^2 + \mu_c^2)(\rho_1 \sigma_1^2 + \mu_1^2) - \mu_3^2}{\sigma_3^2} \tag{33}
\]

where \( \rho_c = \frac{k \sigma_c^2}{\kappa} \) is the correlation coefficient of the sequence of the filter coefficients.

**At the output of the adder, \( y_4(n) \)**

In order to derive the signal sequence at the output of the adder the signals at nodes \( \#2k-\#(3k-1) \) of the filter structure has to be multiplexed according to the proposed method. The statistics of the signals which correspond to the inputs of the multiplexer are given in equation \( (10) \). The statistics for \( y_4(n) \) is expressed as:

\[
\mu_4 = \frac{1}{k} \sum_{i=0}^{k-1} \sum_{j=0}^{k-1} c_i \mu_j
\]

\[
\sigma_4^2 = \frac{1}{k} \left( \sum_{i=0}^{k-1} \sigma_i^2 + \sum_{i=0}^{k-1} \mu_i^2 \right) - \mu_4^2
\tag{34}
\]

where \( \sigma_i^2 \) and \( \mu_i \) corresponds to the variance and the mean value of the signal at the output of the \( i^{th} \) tap of the FIR structure. Finally, the signal at the output of the MAC (at node \( \#5 \)), \( y_5(n) \), is identical with the signal at the output of the filter structure \( y(n) \) in figure 10 and its statistics are also given in equation \( (10) \).

In order to verify our method we use five stationary zero-mean Gaussian input signals with different variances shown in Table 9. The statistics at each node of the MAC unit, which realizes for each of the input signals a 24-tap FIR filter, are measured and compared to the calculated ones according to the proposed method. The coefficients of the filter are given in Table 10. The results for the signal statistics are given in Tables 11-13. As it is observed the calculated results are very close to the measured ones which proves the accuracy the proposed method.

<table>
<thead>
<tr>
<th></th>
<th>( \mu )</th>
<th>( \sigma )</th>
<th>( \rho )</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIG1</td>
<td>-0.482</td>
<td>507.20</td>
<td>0.9511</td>
</tr>
<tr>
<td>SIG2</td>
<td>0.774</td>
<td>498.88</td>
<td>0.6999</td>
</tr>
<tr>
<td>SIG3</td>
<td>-0.393</td>
<td>1000.01</td>
<td>0.7998</td>
</tr>
<tr>
<td>SIG4</td>
<td>-0.163</td>
<td>1034.05</td>
<td>-0.9023</td>
</tr>
<tr>
<td>SIG5</td>
<td>0.421</td>
<td>99.85</td>
<td>-0.0034</td>
</tr>
</tbody>
</table>

**Table 9: Input signals statistics**
Table 10: Coefficients for the 24-tap filter

<table>
<thead>
<tr>
<th></th>
<th>(c_0, c_{23})</th>
<th>(c_1, c_{22})</th>
<th>(c_2, c_{21})</th>
<th>(c_3, c_{19})</th>
<th>(c_4, c_{18})</th>
<th>(c_5, c_{17})</th>
<th>(c_6, c_{16})</th>
<th>(c_7, c_{15})</th>
<th>(c_8, c_{14})</th>
<th>(c_9, c_{13})</th>
<th>(c_{10}, c_{12})</th>
<th>(c_{11}, c_{11})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_0)</td>
<td>0.03390</td>
<td>0.01492</td>
<td>0.01056</td>
<td>-0.01590</td>
<td>-0.03410</td>
<td>-0.03810</td>
<td>0.04010</td>
<td>0.11542</td>
<td>0.18851</td>
<td>0.23355</td>
<td>0.00256</td>
<td></td>
</tr>
</tbody>
</table>

Table 11: Comparison between measured and calculated signal statistics at node #1

<table>
<thead>
<tr>
<th></th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIG1</td>
<td>-0.54</td>
<td>-0.48</td>
<td>11.11</td>
<td>507.20</td>
<td>507.20</td>
<td>0.924</td>
<td>0.924</td>
<td>0.00</td>
<td>0.924</td>
<td>0.924</td>
<td>0.00</td>
<td>0.924</td>
<td>0.924</td>
<td>0.00</td>
<td>0.924</td>
<td>0.924</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIG2</td>
<td>0.76</td>
<td>0.77</td>
<td>1.32</td>
<td>498.90</td>
<td>498.90</td>
<td>0.670</td>
<td>0.670</td>
<td>0.00</td>
<td>0.670</td>
<td>0.670</td>
<td>0.00</td>
<td>0.670</td>
<td>0.670</td>
<td>0.00</td>
<td>0.670</td>
<td>0.670</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIG3</td>
<td>-0.33</td>
<td>-0.39</td>
<td>18.18</td>
<td>999.95</td>
<td>1000.00</td>
<td>0.766</td>
<td>0.766</td>
<td>0.50</td>
<td>0.766</td>
<td>0.766</td>
<td>0.13</td>
<td>0.766</td>
<td>0.766</td>
<td>0.13</td>
<td>0.766</td>
<td>0.766</td>
<td>0.13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIG4</td>
<td>-0.16</td>
<td>-0.16</td>
<td>0.00</td>
<td>1034.00</td>
<td>1034.00</td>
<td>0.861</td>
<td>0.861</td>
<td>0.00</td>
<td>0.861</td>
<td>0.861</td>
<td>0.00</td>
<td>0.861</td>
<td>0.861</td>
<td>0.00</td>
<td>0.861</td>
<td>0.861</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIG5</td>
<td>0.42</td>
<td>0.42</td>
<td>0.00</td>
<td>99.85</td>
<td>99.85</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 12: Comparison between measured and calculated signal statistics at node #3

<table>
<thead>
<tr>
<th></th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
<th>(\mu)</th>
<th>(\sigma)</th>
<th>(\rho)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIG1</td>
<td>0.29</td>
<td>0.25</td>
<td>13.79</td>
<td>351.70</td>
<td>351.70</td>
<td>0.02</td>
<td>0.949</td>
<td>0.948</td>
<td>0.11</td>
<td>0.948</td>
<td>0.948</td>
<td>0.11</td>
<td>0.948</td>
<td>0.948</td>
<td>0.11</td>
<td>0.948</td>
<td>0.948</td>
<td>0.11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIG2</td>
<td>0.41</td>
<td>0.41</td>
<td>23.63</td>
<td>633.50</td>
<td>633.50</td>
<td>0.05</td>
<td>0.945</td>
<td>0.945</td>
<td>0.00</td>
<td>0.945</td>
<td>0.945</td>
<td>0.00</td>
<td>0.945</td>
<td>0.945</td>
<td>0.00</td>
<td>0.945</td>
<td>0.945</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIG3</td>
<td>0.17</td>
<td>0.21</td>
<td>23.53</td>
<td>633.50</td>
<td>633.50</td>
<td>0.09</td>
<td>0.947</td>
<td>0.947</td>
<td>0.00</td>
<td>0.947</td>
<td>0.947</td>
<td>0.00</td>
<td>0.947</td>
<td>0.947</td>
<td>0.00</td>
<td>0.947</td>
<td>0.947</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIG4</td>
<td>-0.96</td>
<td>-0.96</td>
<td>94.40</td>
<td>95.01</td>
<td>95.01</td>
<td>0.65</td>
<td>0.429</td>
<td>0.436</td>
<td>1.63</td>
<td>0.436</td>
<td>0.436</td>
<td>1.63</td>
<td>0.436</td>
<td>0.436</td>
<td>1.63</td>
<td>0.436</td>
<td>0.436</td>
<td>1.63</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIG5</td>
<td>0.22</td>
<td>0.22</td>
<td>33.40</td>
<td>33.40</td>
<td>33.40</td>
<td>0.18</td>
<td>0.920</td>
<td>0.920</td>
<td>0.00</td>
<td>0.920</td>
<td>0.920</td>
<td>0.00</td>
<td>0.920</td>
<td>0.920</td>
<td>0.00</td>
<td>0.920</td>
<td>0.920</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 13: Comparison between measured and calculated signal statistics at node #4

2.6 Calculation of the transition activity

Assuming a Gaussian distribution for the input signals with zero mean value, positive and negative numbers are equiprobable and therefore for the signal probability of the sign bits (two’s complement representation is used here) as well as for the rest bits of a word it holds \(p_0=p_1=0.5\) [80]. Since the values of the signals at the nodes of the MAC architecture present a symmetrical distribution and their mean value is zero, as a multiple of the mean of the input signal, for all bits at all nodes also it holds \(p_0=p_1=0.5\).

At node #1 the resulted signal sequence has almost identical statistics with the filter input signal and is a Gaussian process. Therefore, eqs (15) and (4)-(7) can be used for the calculation of the transition activity at this node. In addition, the transition activity at node #2 can be directly calculated since the filter coefficients are known. Also, since signal values at the output of the filter (node #5) present a normal
distribution, the transition activity at this node can be calculated by the same with the node #1 way. The calculated and measured results for the transition activity at nodes #1 and #5 are given in Table 14. Although the symmetrical signal distributions at nodes #3 and #4 are not normal distributions, the transition activity of the MSB at these nodes can also be calculated from eq.(7) with sufficient accuracy. Moreover, an accurate value for the activity at the MSB of the output node of the multiplier can be derived since the transition activity of the signs at the output of the multiplier can be determined by the transition activity of the sign of its inputs. The calculated and measured results for the transition activity at nodes #3 and #4 are given in Table 15. Finally, in this Table, a comparison between the calculated and the measured total activity in the MAC is also given.

In order to verify the approach followed in section IV for the determination of the breakpoints and the correctness of the transition activity model of eq. (6), the actual and calculated breakpoints for the signals at nodes #3 and #4 are given in Table 16 while the actual and calculated transition activity per bit for SIG2 and SIG4 at nodes #3 and #4 are shown in figure 13 and figure 14, respectively. As it is observed from the results presented in this and the previous section, using the proposed method very accurate estimations of the signal statistics and the transition activity at the nodes of the MAC architecture when it implements FIR filters can be obtained.

<table>
<thead>
<tr>
<th>SIG1</th>
<th>Measure</th>
<th>Estimate</th>
<th>Error %</th>
<th>Measure</th>
<th>Estimate</th>
<th>Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIG2</td>
<td>5.381</td>
<td>5.749</td>
<td>6.84</td>
<td>4.522</td>
<td>5.043</td>
<td>11.52</td>
</tr>
<tr>
<td>SIG3</td>
<td>6.611</td>
<td>6.713</td>
<td>1.54</td>
<td>5.176</td>
<td>5.196</td>
<td>0.39</td>
</tr>
<tr>
<td>SIG4</td>
<td>6.570</td>
<td>6.749</td>
<td>2.72</td>
<td>5.483</td>
<td>5.512</td>
<td>0.53</td>
</tr>
<tr>
<td>SIG5</td>
<td>9.828</td>
<td>9.651</td>
<td>1.80</td>
<td>4.913</td>
<td>4.933</td>
<td>0.41</td>
</tr>
<tr>
<td></td>
<td>8.005</td>
<td>8.008</td>
<td>0.04</td>
<td>4.388</td>
<td>4.474</td>
<td>3.14</td>
</tr>
</tbody>
</table>

Table 14: Comparison between measured and calculated transition activity at nodes #1 and #5

<table>
<thead>
<tr>
<th>SIG1</th>
<th>Measure</th>
<th>Estimate</th>
<th>Error %</th>
<th>Measure</th>
<th>Estimate</th>
<th>Error %</th>
<th>Measure</th>
<th>Estimate</th>
<th>Error %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15.385</td>
<td>15.979</td>
<td>3.86</td>
<td>10.803</td>
<td>10.797</td>
<td>0.06</td>
<td>31.354</td>
<td>31.787</td>
<td>1.38</td>
</tr>
</tbody>
</table>

Table 15: Comparison between measured and calculated transition activity at nodes #3, #4 and the total activity

<table>
<thead>
<tr>
<th>SIG1</th>
<th>BP0</th>
<th>Measure</th>
<th>Estimate</th>
<th>23</th>
<th>22</th>
<th>16</th>
<th>15</th>
<th>25</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIG2</td>
<td>BP0</td>
<td>17</td>
<td>16</td>
<td>23</td>
<td>22</td>
<td>16</td>
<td>15</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>SIG3</td>
<td>BP0</td>
<td>18</td>
<td>17</td>
<td>24</td>
<td>23</td>
<td>17</td>
<td>16</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>SIG4</td>
<td>BP0</td>
<td>18</td>
<td>17</td>
<td>24</td>
<td>23</td>
<td>17</td>
<td>17</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>SIG5</td>
<td>BP0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>14</td>
<td>13</td>
<td>22</td>
<td>22</td>
</tr>
</tbody>
</table>

Table 16: Measured and estimated breakpoints at nodes #3 and #4
Fig. 13: Actual and calculated transition activity per bit for SIG2 and SIG4 at node #3

Fig. 14: Measured and estimated transition activity per bit for SIG2 and SIG4 at node #4
2.7 Conclusions

In this work a method for the calculation of the transition activity on a MAC unit when implements FIR filters has been proposed. The method is based on the estimation of the statistics of the signals which appears at the nodes of the architecture. These signals are considered to be the output of a multiplexer which takes as inputs signals with known statistics. A fully mathematical formulation of this technique is developed and accurate expressions for the signal statistics are derived. Finally, using these statistics the transition activity at the nodes of the architecture is calculated with very good accuracy.

References


