APPLICATION-SPECIFIC ARCHITECTURE FOR FAST TRANSFORMS BASED ON THE SUCCESSIVE DOUBLING METHOD, PART I: A CONSTANT GEOMETRY APPROACH *

Francisco Argüello

Dept. Electrónica y Computación, Facultad de Física,
Universidad de Santiago. 15706 Santiago. Spain

Mailing Address: Francisco Arguello
Dept. Electronica y Computacion
Facultad de Fisica
Universidad de Santiago
15706 Santiago
Spain

PHONE: 34 81 56 31 00 ext. 4043
FAX: 34 81 52 06 76
e-mail: elusive@usc.es

*This work was supported by the Ministry of Education and Science (CICYT) of Spain under contracts TIC 88-0094 and 90-0407, MIC 88-0549 and 90-1264, and Xunta de Galicia XUGA80406488. Published as: Emilio L. Zapata, and Francisco Argüello, *Application-Specific Architecture for Fast Transforms Based on the Successive Doubling Method*, in **IEEE Transactions on Signal Processing**, vol. 41, no. 3, March 1993, pp. 1476–1481.

APPLICATION-SPECIFIC ARCHITECTURE FOR FAST TRANSFORMS BASED ON THE SUCCESSIVE DOUBLING METHOD, PART I: A CONSTANT GEOMETRY APPROACH

Abstract

The successive doubling method is an efficient procedure for the design of fast algorithms for orthogonal transforms of length $N = r^n$, where the radix r is a power of 2. It reduces the algorithmic complexity from N^2 to $N \cdot \log_r N$. In this work we present a partitioned systolic architecture for the two standard radix successive doubling algorithms: ascend and descend communication patterns. The systolization and partitioning procedure we have used is made up of three actions. First, we transform the flow chart of the data for the successive doubling algorithm into a new chart of constant geometry in all its stages (n). We obtain the constant geometry by means of the perfect unshuffle (ascending algorithm) or shuffle (descending algorithm) permutations of order $\log_2 r$. We then carry out the decomposition of these permutations into elementary permutations, which can be implemented electronically. Finally, we project the index space of the data onto the index space associated with a column of processors interconnected using a perfect unshuffle or shuffle interconnection network. The result is a systolic rectangular array with 1 to n columns of Q processors ($Q = r^i,\ 0 \leq i < n$). This architecture extracts the maximum spatial and temporal parallelism achieved by the successive doubling algorithm and can be integrated in VLSI and WSI technologies.

Index Terms: Constant geometry architecture, successive doubling algorithm, systolic design, partitioning, orthogonal transforms, VLSI and WSI technologies.

I INTRODUCTION

The theory of orthogonal transforms has played a key role in the field of multidimensional processing of signals and is still a topic of great interest both from the theoretical and applied point of view. Since computers appeared and use has been made of the possibilities offered by the advances in semiconductor technology, there has been a lot of effort dedicated to reducing the calculation time and/or the memory needs of the transforms. Efforts which have been aimed both at the design of faster and/or more efficient numerical algorithms for conventional computers and at the exploitation of the possibilities offered by parallel machines (extraction of the inherent concurrence) and the new advances in the design of application specific integrated circuits (ASIC).

In 1965 Cooley and Tukey [19] developed an algorithm for accelerating the calculation of the discrete Fourier transform (DFT) which in some ways revolutionized the numerical computation of orthogonal transforms. This algorithm, known as the Fast Fourier Transform (FFT) radix 2 and ascend communication pattern, is based on the application of the method of successive doubling for the elimination of the inherent redundances in the coefficient matrix of the DFT transform. Since the discovery of the FFT algorithm there have been considerable efforts dedicated to improving it and to extend its application range. A first step was to transform the initial recursive algorithm into a more efficient one with a new structure of nested loop indexing. Later, the radix was increased to reduce the necessary arithmetic. Bergland [7] and Sande [69] developed algorithms for the calculation of FFTs in real sequences (RFFT). Winograd applied the theory of computational complexity to the calculation of the DFT [85] obtaining this way a lower limit for the number of multiplications required for the computation of a DFT of 2^n elements and designed a constructive method for the generation of these algorithms.

Burrus [16] extended the application of the DFT to sequences whose length is the product of two relative prime factors (PFFT). More recently, algorithms [21,74,82,88] known as split-radix FFT (SRFFT), which have an optimum number of multiplications and the minimum known number of additions have been [34] developed. A unified set of algorithms which define the interconnection and the rotation structure of the phase of the flowchart of arbitrary FFTs have been recently developed by Demuth [20]. Finally, the large number of books and articles that have been appearing are a required reference for analyzing and understanding the evolution of the aforementioned algorithms [11,15,40,59,64-65].

In general, system architecture has been greatly influenced by the advances in the technological processes of microelectronics, constantly requiring new ideas for the organization of processing [17,76]. The most important advantages currently offered by VLSI (very large scale integration) and WSI (Wafer Scale Integration) technology are a reduced physical size, with a low power consumption at a really low cost and the possibility of eliminating the need of processors which are physically separated from the memory or from other processors. The most important disadvantage they present, WSI in particular, is the need of introducing redundances [89-90,36] and/or fault tolerance [5,45,77] in the designs because the circuits are integrated over semiconductor areas which are sufficiently large to make the appearance of defects unavoidable. A solution for minimizing the redundant silicon area [89] consists in designing architectures with a regular structure and, where possible, with parallel logic for the following reasons: a) easy interconnection of active circuit blocks; and b) the global performance of the system is better when the number of parallel operations is increased, as a consequence of having a large number of identical devices in the wafer.

Not all the fast algorithms for the FFT transform we have mentioned permit an immediate implementation using VLSI and WSI technologies. Some of them do not have the necessary regularity in the data flow to make them integrable. This is the case of the Split Radix SRFFT algorithm, which has a different number of butterflies for each stage of the transform [66]. The PFFT [18,31,80] algorithms present more regularity although they have a higher complexity than the equivalent FFT. The idoneous candidate for integration is the FFT algorithm, as we can achieve constant indexing throughout all the stages of the transform [64], obtaining a regular structure and a simpler control. The result is a constant geometry algorithm which permits the exploitation of the spatial parallelism presented by the FFT algorithm [61].

Parallel to the development of fast algorithms for the discrete Fourier transform, there has been a development of similar algorithms and architecture proposals for other transforms, such as those of Walsh [30], Hartley [14,32], Haar [67] and cosine [2]. Our objective in this work is to design an application specific architecture which permits the exploitation of the parallelism present in the successive doubling method and which is integrable in VLSI and WSI technologies. In a companion paper [93] we describe the specific designs of the six fast transforms we have considered: Walsh, complex valued Fourier, Hartley, real valued Fourier, cosine and Haar. Only the first two present a data flow coinciding with the flow chart of the successive doubling algorithm. The other four require some type of additional transformation in order to have this flow chart. Moreover, each transform will have a different processing section.

An adequate architecture for integration in VLSI and WSI technologies is the systolic architecture, proposed by Kung and Leiserson [47]. The projection of an algorithm onto a systolic architecture requires the performance of transformations (regularization stage) which extract the spatial parallelism of the algorithm [24]. It is also necessary to approach the partitioning of the algorithm in order to facilitate integration. There are a lot of methods for the systolization and partitioning of algorithms [24,55-57], but none of them is applicable to the algorithms based on the successive doubling method. This is the reason why a large amount of authors consider the FFT algorithm as non systolizable and, consequently, they have directed their research towards the design of systolic architectures for the discrete transforms [4,9-10,12,25-27,53,71,81,87], or the design of application specific microprogrammable processors [3,29,37-38,51]. On the other hand, we must point out that all the algorithms for the fast transforms are easily implementable in those processors whose base is a multiplier/accumulator, being the most efficient those algorithms which minimize the number of multiplications and/or additions. This is the case of the ones known as Digital Signal Processors (DSPs) [1,22-23,48-50,79], which do not exploit the spatial parallelism of the algorithms.

We have structured the rest of this paper in the following way. The successive doubling method is introduced in section II, defining the two standard algorithms established by the direction we follow through the flow chart [64]: ascend communication pattern (ACP algorithm) and descend communication pattern (DCP algorithm). In sections III and IV we present in detail the application specific architecture of the DC algorithm, defined by the perfect unshuffle permutation. More specifically, in section III we approach the design of the appropriate processor for obtaining the constant geometry systolic architecture and in section IV we present the application specific parallel architecture. In section V we briefly describe the application specific architecture associated with the DCP algorithm, whose constant geometry is determined by the perfect shuffle permutation.

II THE SUCCESSIVE DOUBLING METHOD

The discrete Fourier transform belongs to a class of important transforms which can be expressed in terms of the general relation

$$X(k) = \sum_{m=0}^{N-1} T(k,m) \cdot x(m) \tag{1}$$

where $X(k)$ is the transform of $x(m)$, $T(k,m)$ is the kernel of the direct transform, and k is a variable which takes values in the range $0, 1, \ldots, N-1$. In a similar way, the inverse transform is defined by the relation

$$x(m) = \sum_{k=0}^{N-1} T^{-1}(m,k) \cdot X(k) \tag{2}$$

where $T^{-1}(m,k)$ is the inverse transform kernel and m is a variable which takes values in the range $0, 1, \ldots, N-1$. Belonging to this class of transforms we have the discrete transforms of Walsh [30], Hartley [14,32], Haar [67] and cosine [2], among others. In general, the nature of a transform is determined by the properties of its transformation kernel.

Fourier:

$$T(k,m) = \exp\left(-j2\pi \frac{mk}{N}\right) \tag{3}$$

Walsh:

$$T(k,m) = \prod_{i=0}^{n-1} (-1)^{m_i k_{n-i-1}} \tag{4}$$

Hartley:

$$T(k,m) = \cos\left(2\pi \frac{mk}{N}\right) + \sin\left(2\pi \frac{mk}{N}\right) \tag{5}$$

Cosine:

$$T(k,m) = e(k) \cdot \cos\left(\pi \frac{(2m+1)k}{2N}\right) \tag{6}$$

Haar:

$$T(k,m) = \begin{cases} 2^{p/2} & s2^{-p} \leq m \leq (s+1/2)2^{-p} \\ -2^{p/2} & (s+1/2)2^{-p} < m \leq (s+1)2^{-p} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where b_i is the i-th bit of the binary representation of b. For the discrete cosine transform $e(0) = 1/\sqrt{2}$ y $e(k) = 1$, $0 < k < N$; and for the discrete Haar transform $k = 2^p + s$, being $0 \leq p < \log_2 N$ and $0 \leq s < 2^p$.

These transforms can be calculated using a fast algorithm by applying the successive doubling method, the objective of which is to minimize the redundant operations. The idea of successive doubling, used by Cooley-Tukey for the design of their fast Fourier algorithm [19], consists in dividing the original N element sequence $x(m)$ into two sequences of half the length. The discrete transforms of these have to be combined to obtain the discrete transform $X(k)$ of the original sequence. The successive doubling method consists in performing successive bisections of the data until the original sequence is decomposed into N/r sequences of length r ($N = r^n$), where r is the length of the minimum sequence to be transformed (radix of the transform). Once the N/r discrete transforms (butterflies) have been calculated, they must be combined to obtain the discrete transform of the original sequence by means of $\log_r N$ calculation stages.

The direct evaluation of equations (1) and (2) presents an algorithmic complexity $O(N^2)$, which is reduced for orthogonal transforms to $O(N \cdot \log_r N)$ if we apply the successive doubling method. Figure 1 shows the data flow for the radix 2 fast transform of a sequence of $N = 16$ elements, using the successive doubling method. The sequence, of length 16, is decomposed into 8 ($N/2$) elementary sequences of length 2 ($r = 2$), whose discrete transforms are combined in four stages in order to obtain the transform of the original sequence. The data flow of the figure can be seen from left to right (ascend communication pattern algorithm, ACP) or from right to left (descend communication pattern algorithm, DCP). As a result of the successive bisections of the initial data set it is necessary to carry out a shuffle of the input sequence (ACP algorithm) or of the transformed sequence (DCP algorithm) in order to obtain an output sequence ($X(k)$, $0 \leq k < N$) in its natural order. The usual way for carrying out this shuffle is by using the bit reversal permutation.

From the analysis of figure 1 we can extract three conclusions. The first one is that the data flow between stages is not constant. However, by reordering the butterflies from each stage we can produce a constant geometry fast transform [64,33] such as the one in figure 2. The second one is the high inherent parallelism in each stage of the transform (N/r butterflies in parallel), which we can exploit

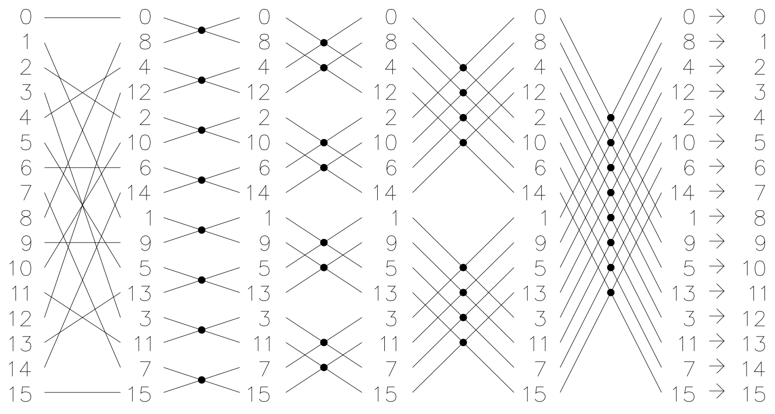

Figure 1: Data flow for Cooley Tukey's algorithm ($N = 16$).

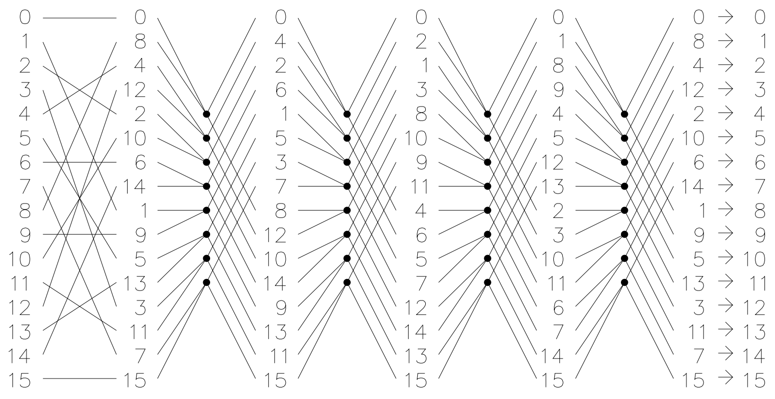

Figure 2: Cooley-Tukey's constant geometry algorithm ($N = 16$).

if we have a processor column (N/r in the optimum case). The third one is the inherent sequentiality between stages of the transform, which will allow only pipelined designs between stages.

It can be easily seen that the constant geometry of figure 2 is the result of applying a permutation to the results of each stage of the transform. In general, for the ACP algorithm, the appropriate permutation is a perfect unshuffle of order $\log_2 r$ ($r = 2$ in figures 1 and 2), whereas for the DCP algorithm the permutation must be a perfect shuffle of order $\log_2 r$. As both algorithms have the same characteristics in sections III and IV we will describe in detail the application specific architecture associated with the ACP algorithm and in section V we will summarize the equivalent architecture for the DCP algorithm. We will assume that the bit reversal permutation of the input sequence has already been carried out for the input sequence of the ACP algorithm.

Two have been the most used approaches for mapping the inherent parallelism in the data flow of figure 1 onto an architecture which can be implemented in VLSI or WSI technology: pipeline and array. The design of pipelined processors

(PE) [8,54,64,70,72,86,76] is simple but their greatest limitation is their small I/O bandwidth. This limitation can be avoided, in part, by increasing the radix of the transform. With a column of PEs forming a constant geometry architecture we can exploit the spatial parallelism existing in each stage. There are numerous interconnection networks for the PE column which permit an efficient use of this spatial parallelism in each stage: shuffle exchange [75,72,60,89,36], shift and replace [68], hypercube [35,44,92], indirect binary n-cube [62], cube-connected cycles [63], mesh [44,52], among others [6,13,28,39,46,73,78,84,42-43,58]. We can combine both approaches by constructing a rectangular array made up of PE column pipelines [72]. Consequently, the appropriate architecture for the algorithms based on the successive doubling method will be a rectangular array made up of $\log_r N$ columns of N/r processors, connected so as they implement the data flow of figure 2.

The translation to an ASIC of the rectangular array requires a systolic type data flow and, more important still, the problem of partitioning the algorithms in order to design non restrictive systems has to be approached. In this line, Zapata et al. [92] have recently designed SIMD algorithms for the FFT transform in hypercube computers with a limited number of PEs. However, the hypercube topology is not the most appropriate for implementation in VLSI technology, as it has a high number of links compared to some of the networks we have mentioned. You and Wong [91] have also proposed an architecture based on the r-fold symmetry in the radix r constant geometry FFT algorithm, which requires a microprogrammed data shuffler in each one of the processors and limits to r PEs the maximum parallelism of each stage of the transform. The constant geometry architecture we present in the following sections is based on the perfect unshuffle interconnection network, which permits efficient mapping and partitioning of the flow chart generated by the successive doubling method without having to use microprogrammed control. This architecture can be considered semisystolic: regular with systolic type data flow, but the connectivity between nodes is not local.

III THE CONSTANT GEOMETRY ARCHITECTURE

In order to express the design with constant geometry in a general way, we will use the notation introduced by Parker [60] for the definition of a set of algebraic operators which permit the description of processor networks in terms of their interconnection rules. This operators are associated with the different bit permutations which can be carried out on the binary representation of the numbers. We will center on those permutations which allow us to implement successive doubling algorithm in a constant geometry parallel architecture.

We consider that the size of the transform is $N = r^n$, where r is the radix and we will use a two dimensional representation $[x, z]$ of the index for each data item $(i = 0, 1, \ldots, N-1)$ in the input sequence

$$[x, z] = [[x_u \cdots x_1], [z_v \cdots z_1]] \tag{8}$$

where x_j and z_j are the digits of the binary representation of x and z, respectively. The union of x and z into one number $(x \cdot 2^v + z)$ will coincide with the binary representation of the index i of the data sequence $(u + v = \log_2 N)$. Finally, we will suppose that the data sequence flows from left to right. This implies that x counts from right to left so that the first data item which enters from the left will have an index x which is equal to 0 and z counts from top to bottom. Therefore, the original one dimensional data sequence (one N column row) starts with a data item with an index $[0, 0]$ and ends with a data item with an index $[N-1, 0]$.

The operators are defined by their effect on the indexes of the data items. The decimation operator $\delta_{(k)}$, introduced by Wold and Despain [86] converts a row into many by reducing the number of columns

$$\delta_{(k)}[x, z] = [[x_u \cdots x_{k+1}], [z_v \cdots z_1 x_k \cdots x_1]] \tag{9}$$

Each row is broken into 2^k rows, and the operator is well defined if $k \leq u$. As an example of the operation of $\delta_{(k)}$, consider a row of data items with $u = 3$ and $v = 0$. The data enter from the left and are given by

$$(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0) \tag{10}$$

By applying the operator $\delta_{(1)}$ to the indexes this sequence is converted into a two dimensional array of size 2 by 4

$$(a_7\ a_6\ a_5\ a_4\ a_3\ a_2\ a_1\ a_0) \xrightarrow{\delta_{(1)}} \begin{pmatrix} a_6 & a_4 & a_2 & a_0 \\ a_7 & a_5 & a_3 & a_1 \end{pmatrix} \tag{11}$$

Using this notation, we can define the operator concatenation $\beta_{(k)}$ which reduces the number of rows of an array by increasing the number of columns

$$\beta_{(k)}[x, z] = [[z_k \cdots z_1 x_u \cdots x_1], [z_v \cdots z_{k+1}]] \tag{12}$$

A sequence made of 2^v rows with 2^u elements (columns) is transformed into another with 2^{v-k} rows of 2^{u+k} columns and the operator is well defined if $k \leq v$. Let's consider a sequence formed by two rows of data which flow from left to right and which is similar to the one generated by the operator $\delta_{(1)}$ $(u = 2,\ v = 1)$.

$$\begin{pmatrix} a_6 & a_4 & a_2 & a_0 \\ a_7 & a_5 & a_3 & a_1 \end{pmatrix} \xrightarrow{\beta_{(1)}} (a_7\ a_5\ a_3\ a_1\ a_6\ a_4\ a_2\ a_0) \tag{13}$$

The application of the operator $\beta_{(1)}$ generates a one dimensional sequence by concatenating the two input rows.

Finally, we define the perfect unshuffle operator $\Gamma_{(k)}$ of a sequence, which flows from left to right and is organized as a two dimensional array of 2^v rows and 2^u columns

$$\Gamma_{(k)}[x,z] = [[z_k \cdots z_1 x_u \cdots x_{k+1}],[x_k \cdots x_1 z_v \cdots z_{k+1}]] \tag{14}$$

$\Gamma_{(k)}$ performs a rotation to the right of order k of the binary representation of the index of each element of the sequence and is well defined if $k \leq u+v$.

We are interested in the efficient hardware implementation of the perfect unshuffle permutation as it is the base for the design of a constant geometry architecture for radix r ACP successive doubling algorithm. The output sequence of the processor will have to undergo a perfect unshuffle permutation in order to maintain the constant geometry in all stages of the transform. From this we deduce that out of all the permutations we can implement with equation (9) only the particular cases $\Gamma_{(v)}[x,z]$ and $\Gamma_{(v)}[x,[]]$ will be of any interest, being $v = \log_2 r$. Also, this permutations can be obtained by the combination of the operators $\delta_{(v)}$ and $\beta_{(v)}$, defined previously.

Lemma 1

$$\begin{aligned} \Gamma_{(v)} &= \beta_{(v)}\delta_{(v)}, \;\; v \neq 0 & (15) \\ \Gamma_{(i)}[x,[]] &= \delta_{(i)}\beta_{(i)}[x,[]], \quad i \neq u & (16) \end{aligned}$$

Being the order for the application of the operators from left to right.

Proof Proof of (15):

$$\begin{aligned} \beta_{(v)}\delta_{(v)}[[x_u \cdots x_1],[z_v \cdots z_1]] &= \delta_{(v)}[[z_v \cdots z_1 x_u \cdots x_1],[]] \\ &= [z_v \cdots z_1 x_u \cdots x_{v+1}],[x_v \cdots x_1]] \\ &= \Gamma_{(v)}[[x_u \cdots x_1],[z_v \cdots z_1]] \end{aligned} \tag{17}$$

Proof of (16):

$$\begin{aligned} \delta_{(i)}\beta_{(i)}[[x_u \cdots x_1],[]] &= \beta_{(i)}[[x_u \cdots x_{i+1}],[x_i \cdots x_1]] \\ &= [x_i \cdots x_1 x_u \cdots x_{i+1}],[]] \\ &= \Gamma_{(i)}[[x_u \cdots x_1],[]] \end{aligned} \tag{18}$$

□

As an example, observe that the output generated in (13) coincides with the perfect unshuffle permutation $\Gamma_{(1)}$ of the original sequence (10) (particular case

$[x, []]$ with $u = 3$). The output in (13) is the result of applying the decimation ($\delta_{(1)}$ generates the output sequence (11)) and concatenation ($\beta_{(1)}$ generates the output sequence (13)) permutations to sequence (10). In a similar way, if we perform the permutation $\delta_{(1)}$ on the output sequence expressed in (13)

$$(a_7\ a_5\ a_3\ a_1\ a_6\ a_4\ a_2\ a_0) \xrightarrow{\delta_{(1)}} \begin{pmatrix} a_5\ a_1\ a_4\ a_0 \\ a_7\ a_3\ a_6\ a_2 \end{pmatrix} \tag{19}$$

we obtain the perfect unshuffle permutation $\Gamma_{(1)}$ of the original sequence which acts as input in (13) (particular case $[x, z]$, with $u = 2$ and $v = 1$). Observe that in (13) we have applied the concatenation permutation $\beta_{(1)}$.

A Design of the processor

The internal structure of the processor will consist of two clearly differentiated sections: Processing (PS) and routing (RS). The PS section will carry out the set of operations associated with a r-point butterfly (discrete transform of a sequence with r-points). This operations will depend on which particular transform we are implementing. As we are only interested in the regrouping of the data items and not in the specific computations of each transform, we will consider the PS section as a new operator, the butterfly operator $B_{(v)}$, which carries out an arbitrary function with 2^v-inputs and 2^v-outputs.

The equalities (15) and (16) guarantee the decomposition of the perfect unshuffle permutation into two elementary permutations which are easily implemented in hardware. Specifically, the concatenation permutation $\beta_{(v)}$ can be implemented using a FIFO queue of length N with 2^v inputs located in cells 0-, 2^u-, ..., and $(2^v - 1) \cdot 2^u$-th, using a numbering scheme from left to right; the queue must have an output in cell $N - 1$.

There are two ways of implementing the decimation permutation $\delta_{(v)}$. The first can be achieved by means of a FIFO queue of length N cells ($i = 0, 1, \ldots, N-1$, $N = r^n$) with outputs in the cells $(N - 1)$-, $(N - 2)$-, ... , and $(N - 2^v)$-th considering the same numbering scheme as in the previous case; the queue must have an input in cell zero. We can also implement the permutation $\delta_{(v)}$ by means of a demultiplexor with an input associated with the sequence we wish to decimate, 2^v outputs and v control inputs used in a cyclic fashion each clock period. Both solutions require the sequence to be decimated to advance 2^v positions each cycle.

The hardware implementation of permutation $\Gamma_{(v)}$ is immediate using permutation $\beta_{(v)}$ and one of the two alternatives of permutation $\delta_{(v)}$. This possibility of choosing produces two different designs for the processor, although the internal parallelism is the same in both cases (2^v data items are processed in parallel). If we use the FIFO queue as the implementation for operator $\delta_{(v)}$ (Lemma 1, equality (15)), the design of the processor for the calculation of a stage of ACP

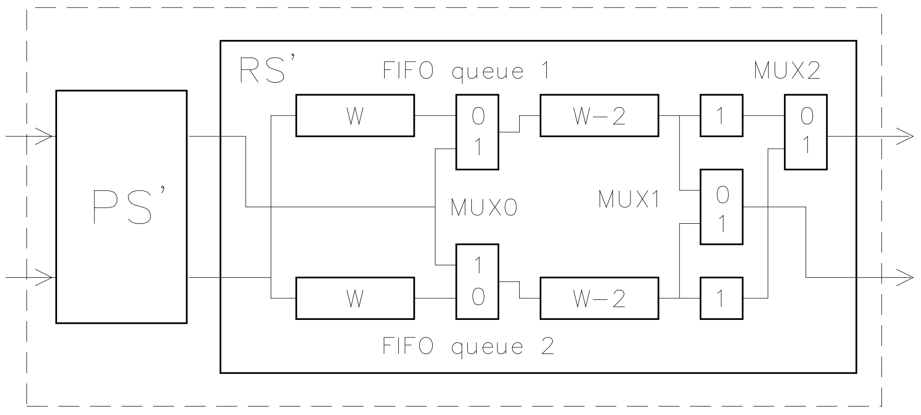

Figure 3: PE based on the operator string $B_{(v)}\beta_{(v)}\delta_{(v)}$ (algorithm ACP).

successive doubling algorithm is the hardware translation of the following operator string

$$B_{(v)}\beta_{(v)}\delta_{(v)} \tag{20}$$

being $v = \log_2 r$. Figure 3 shows the design of the radix 2 processor. We have included a double FIFO queue in order to be able to implement the whole transform by external recirculation of the data, using only one processor with the i-th output connected to the i-th input ($i = 0, 1, \ldots, 2^v - 1$). The n stages ($n = \log_r N$) of the transform are identical, as we apply the operator sequence (20) n times. Each stage, a queue acts as the output buffer (writing the data generated in the current stage) whereas the other acts as input buffer (reading the data generated in the previous stage) and this function will be exchanged in the next stage, this operation is controlled by multiplexors $MUX0 - MUX2$. Observe that in the design of figure 3 the FIFO queues have a length of $N - 1$ cells ($W = N/r$) and the inputs associated with permutation $\beta_{(v)}$ have been conveniently distributed. In order to do this we have considered the PS section as a segment of the pipeline made of sections PS and RS.

Figure 4 shows the second alternative for the design of the processor (Lemma 1, equality (16)), we have considered radix 2 again. This design is the hardware translation of the following operator string

$$\delta_{(v)}B_{(v)}\beta_{(v)} \tag{21}$$

where $v = \log_2 r$. For the same reasons as in the design of figure 3 we have included two FIFO queues of lengths $N - 1$ with 2^v inputs and only one output in its left end (cell ($N - 2$)-th). In this case we can also implement all the stages of ACP successive doubling algorithm using only one processor, with each output bus feeding back its corresponding input bus.

The design of the processor according to the operator sequences (20) or (21)

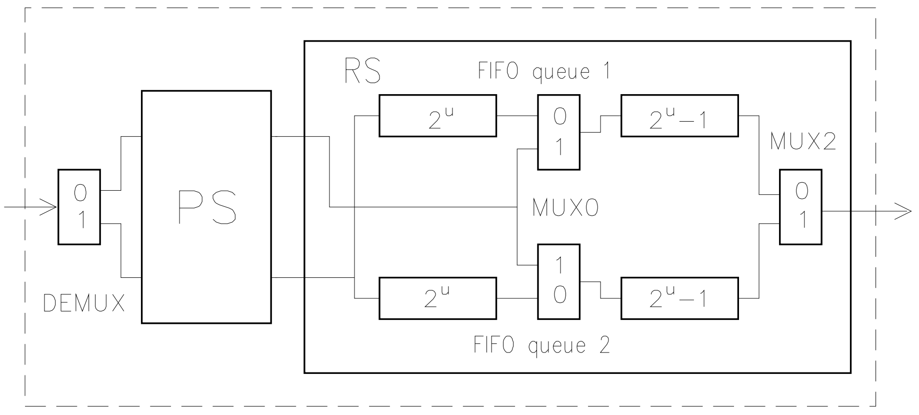

Figure 4: PE based on the operator string $\delta_{(v)}B_{(v)}\beta_{(v)}$.

permits the interpretation of the two dimensional representation of equation (8) in the following way: the z coordinate gives the parallelism for each stage of the transform (a butterfly with 2^v data items is processed each cycle), whereas the x coordinate establishes the sequentiality for each stage of the transform (2^u butterflies of length 2^v). Therefore, the calculation time for a stage will be 2^u clock cycles, being the length of the cycle the time used by the processor in the computation of the butterflies associated with each input vector. With this interpretation of equation (8), the binary representation of the data consists of two fields $[cycle, bus]$. The data item $[x, z]$ will input the processor through its z-th input bus ($bus = 0, 1, \ldots, 2^v - 1$), being a part of the x-th butterfly of the stage ($cycle = 0, 1, \ldots, 2^u - 1$).

Both processor designs share many common properties: by means of the feedback of the output buss with the corresponding input buss, the processor evaluates the whole transform; they have the same processing speed (2^v data items each cycle); they need the same number of memory cells ($2(N-1)$); the information in the FIFO queues advances 2^v cells each cycle; the PS section is identical. Nevertheless, they present some important differences which are a consequence of way of implementing permutation $\Gamma_{(v)}$ (operator strings (20) and (21)): The RS section of the design in figure 4 is divided into two blocks (DEMUX and FIFO queues) and, even more important, it only has one input bus and one output bus for the flow of data, whereas the design of figure 3 requires 2^v input buss and 2^v output buss.

From what has been said, we could deduce at first sight that the design of figure 3 is a lot less efficient than that of figure 4, as it has the same processing speed with a larger amount of input and output buss. Nevertheless, as we will see in the next section, the design of figure 3 will permit a parallel organization with multiple processors operating in array mode (spatial parallelism) whereas the design of figure 4 is only useful in designs with only one processor or multiple processors connected in pipeline mode.

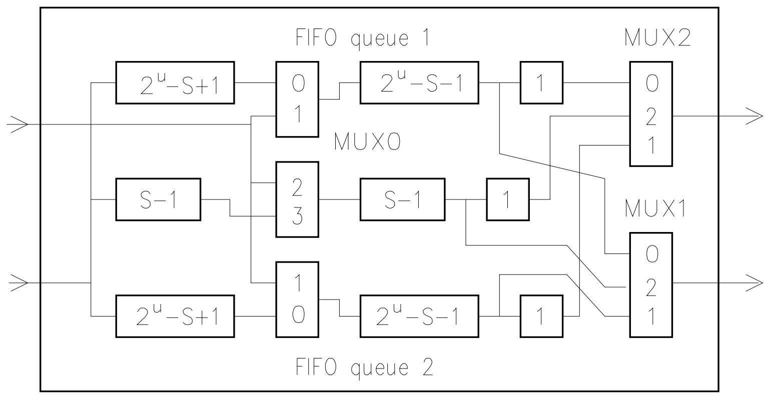

Figure 5: RS section considering S segments in the PS section (radix 2).

Before ending this section we will comment some aspects related with the duration of the processing cycle of the PE. The length of the minimum cycle is given by the slowest route of the processor, which will be determined by the processing time of the PS section or by the number of shifts to the right (r) of the FIFO queues for each butterfly. Therefore, if we consider the PS section as the slowest element of the PE, a systolic or pipelined design of this stage seems the most appropriate solution from the viewpoint of integration in VLSI technology.

In general, the inclusion of an specific number of segments $(S \geq 1)$ in the PS section reduces the length of the operating cycle $(1/S$ factor) and forces, at the same time, if the architecture is not modified, the inclusion of $S-1$ waiting cycles at the end of each stage of the transform. The modification of the architecture of the PE will consist in reducing the length of the FIFO queues by $(S-1) \cdot 2^v$ cells in order to consider the S segments of the PS section as their extensions, achieving the desired overlap between the stages of the transform.

Figure 5 shows the new RS section, particular case of radix 2. The length of the FIFO queues is now $N-(S-1) \cdot 2^v - 1$ cells (2^v segments of length $2^u - S + 1$), but we have had to introduce an additional queue of length $(S-1) \cdot 2^v$ cells distributed in 2^v segments. With this modification we solve the problem of cycle loss. Its operating mode is simple. The main FIFO queue starts unloading in the usual way at a speed of 2^v words per cycle, the additional queue will store the last $S-1$ butterflies of each stage which have to be loaded at the normal speed. Once the main queue has been unloaded, the reading of the additional queue will be performed to complete the stage. This reading will be controlled by the new extended multiplexors $MUX1$ and $MUX2$. Observe that this additional queue is only used for a short interval of time $(S-1$ cycles), and it can be used by both FIFO queues, this is the reason for only including one in the design of the new RS section. Concluding, the pipeline design of the PS section reduces the duration of the processing cycle and reduces the memory requirements in the RS section by $(S-1) \cdot 2^v$ cells, although it increases its complexity.

IV PARALLEL ARCHITECTURE

Successive doubling algorithm can be implemented on a rectangular constant geometry array of PEs. Two extremes of this array are the row of n PEs and the column with $Q = r^q$ PEs, where $n = \log_r N$ and $0 \leq q \leq n-1$. The row of PEs permits pipeline design, the biggest problems of which are the reduced I/O bandwidth as a consequence of the necessary limitation in the number of input and output buss of each PE and the limitation in the number of PEs ($\log_r N$). Its most important advantage is that it permits the sequencing of the transforms without losing cycles. In the other extreme, the column of PEs impedes the execution of a new transform until the current one has finished, but it permits the extension of the I/O bandwidth directly, as the limitation in the number of PEs (N/r) is less restrictive than in the case of the row of PEs. Also, the longer the column the shorter the FIFO queues.

A Processor column: Partitioning

We have just pointed out that the length of the PE columns does not necessarily have to be equal to the number of butterflies of each stage (N/r). This leads to the problem of partitioning successive doubling algorithm for parallel processing. Thus, in what follows, we will consider one column of length Q where $Q \leq N/r$.

To change from a single PE system to a PE column will force us to modify the notation introduced at the beginning of section III as we need a three dimensional representation $[x, y, z]$ of the index of each data item in the input sequence

$$[x, y, z] = [[x_u \cdots x_1], [y_w \cdots y_1], [z_v \cdots z_1]] \tag{22}$$

Where x_j, y_j and z_j have a similar meaning to the one in equation (8) and the union of x, y, and z into a single number $(x \cdot 2^{w+v} + y \cdot 2^v + z)$ will coincide with the binary representation of the index of the data sequence ($i = 0, 1, \ldots, 2^{u+w+v} - 1$; $u + w + v = n$).

If we consider $v = \log_2 r$ we can interpret the three dimensional representation of equation (20) in the following way: the y and z coordinates determine the parallelism of each stage of the transform, 2^w butterflies ($2^w \leq N/r$) of 2^v data items are computed in parallel in one column of 2^w PEs with 2^v inputs each, and the x coordinate establishes the sequentiality in each stage of the transform (2^u vector of length 2^{w+v}). The calculation time for a stage will be 2^u clock cycles where the duration of the cycle is the time used by the processor in the computation of a butterfly of 2^v data items.

With this interpretation of equation (22) we are decomposing the binary representation of the index of each data item into three fields $[cycle, PE, bus]$. In each stage, cycle indicates the instant it is processed ($cycle = 0, 1, \ldots, 2^u - 1$), PE indicates the PE where it will be processed ($PE = 0, 1, \ldots, 2^w - 1$) and bus specifies the bus through which it will enter the PE ($bus = 0, 1, \ldots, 2^v - 1$). Thus, for example, let $N = 64$, $Q = 4$ and $r = 2$ ($(u, w, v) = (3, 2, 1)$) In the first stage of the transform, data item 35 (100011 binary), whose three dimensional representation is [[100],[01],[1]], will input PE 1 through bus 1 in cycle 4 (35 will be a part of the fifth block of butterflies).

The perfect unshuffle permutation of order k on the three dimensional representation of the data indexes is defined as

$$\Gamma_{(k)}[x, y, z] = [[z_k \cdots z_1 x_u \cdots x_{k+1}], [x_k \cdots x_1 y_w \cdots y_{k+1}], [y_k \cdots y_1 z_v \cdots z_{k+1}]] \tag{23}$$

where $k \leq u+w+v$. We are again interested in the decomposition of permutation $\Gamma_{(v)}$ into elementary permutations. For this reason we are going to generalize $\Gamma_{(k)}$, in order to be able to apply it to one, two or the three dimensions of equation (22).

$$\Gamma^{x,z}_{(k)}[x, y, z] = [[z_k \cdots z_1 x_u \cdots x_{k+1}], [y_w \cdots y_1], [x_k \cdots x_1 z_v \cdots z_{k+1}]] \tag{24}$$

$$\Gamma^{z}_{(k)}[x, y, z] = [[x_u \cdots x_1], [y_w \cdots y_1], [z_k \cdots z_1 z_v \cdots z_{k+1}]] \tag{25}$$

where $k \leq v + u$ and $k \leq v$, respectively. We define the rest of the permutations of two and one variables $\Gamma^{a,b}_{(k)}$ and $\Gamma^{c}_{(k)}$, with $a, b = x, y, z$ ($a \neq b$) and $c = x, y, z$ in a similar way. We will also extend the meaning of the operators $\delta_{(k)}$ and $\beta_{(k)}$, which possess a two dimensional nature, in order to be able to apply them to the new representation of equation (22). This is, $\delta^{x,z}_{(k)}$ and $\beta^{x,z}_{(k)}$ perform the decimation and concatenation permutations, equations (9) and (12) respectively, on the dimensions x and y without modifying dimension z.

Lemma 2

$$\Gamma_{(v)} = \Gamma^{x,z}_{(v)} \Gamma^{y,z}_{(v)} \tag{26}$$

Where the application order for the operators is from left to right.

Proof

$$\begin{aligned} & \Gamma^{x,z}_{(v)} \Gamma^{y,z}_{(v)}[[x_u \cdots x_1], [y_w \cdots y_1], [z_v \cdots z_1]] \\ & \quad = \Gamma^{y,z}_{(v)}[[z_v \cdots z_1 x_u \cdots x_{v+1}], [y_w \cdots y_1], [x_v \cdots x_1] \\ & \quad = [[z_v \cdots z_1 x_u \cdots x_{v+1}], [x_v \cdots x_1 y_w \cdots y_{v+1}], [y_v \cdots y_1]] \\ & \quad = \Gamma_{(v)}[[x_u \cdots x_1], [y_w \cdots y_1], [z_v \cdots z_1]] \end{aligned} \tag{27}$$

□

Lemmas 1 and 2 guarantee the decomposition of permutation $\Gamma_{(v)}$ of a two or three dimensional representation of the index of each data item into more elementary permutations, which is the base for the design of a constant geometry architecture. Consequently, we can state the following theorem

Theorem 1 *ACP successive doubling radix r and constant geometry algorithm of a sequence of N data items ($N = r^n$) can be carried out in a column of Q PEs ($Q = r^q$, $0 \leq q \leq n-1$) which implements (hardware translation) each stage the following operator string*

$$B_{(v)}\beta_{(v)}^{cycle,bus}\delta_{(v)}^{cycle,bus}\Gamma_{(v)}^{PE,bus} \tag{28}$$

where $v = \log_2 r$, the operators are applied from left to right and we consider the $[cycle, PE, bus]$ interpretation of the three dimensional representation of the index of each data item.

Proof

$$\begin{aligned} \Gamma_{(v)} &= \Gamma_{(v)}^{x,z}\Gamma_{(v)}^{y,z} \\ &= \beta_{(v)}^{x,z}\delta_{(v)}^{x,z}\Gamma_{(v)}^{y,z} \end{aligned} \tag{29}$$

□

Figure 6 shows the three elements $[cycle, PE, bus]$ for each stage of the radix 2 transform of a sequence of 64 data items in a column with 8 PEs ($N = 64$, $Q = 8$, $r = 2$). We have applied the operator string (26) to each data item each stage. Remember that the sequence to be transformed has been shuffled in accordance with the bit reversal permutation before starting the transform. As was to be expected, the output stage presents the same order as the input sequence, due to the fact that we have carried out six ($n = 6$) permutations $\Gamma_{(1)}$.

The hardware implementation of the perfect unshuffle permutation expressed in (23) is immediate from the operator strings in (26) and (28). $\beta_{(v)}^{cycle,bus}\delta_{(v)}^{cycle,bus}$ performs the perfect unshuffle permutation of the 2^u butterflies of 2^v data items processed sequentially by each PE. Consequently, we can use the same solution as in the single processor case (see (20) and figure 3), with the only difference that in this case the length of the FIFO queues will be $2^u - 2^v$ cells, considering the PS section as the only stage of the internal pipeline of the PE. $\Gamma_{(v)}^{PE,bus}$ performs the perfect unshuffle permutation of the outputs of the PEs. Its hardware implementation will be using an external interconnection network determined by operator $\Gamma_{(v)}^{PE,bus}$ applied on the dimensions $[y, z]$: The z-th output bus of the y-th PE will be connected to the z^*-th input bus of the y^*-th PE, where $[x, y^*, z^*] = \Gamma_{(v)}^{y,z}[x, y, z]$. Figure 7 shows the connections of the PEs for the example of figure 6. The number of input and output buss of each PE is only a function of the radix of the transform and not of the number of PEs in the column.

Summarizing, we have introduced the partition of ACP successive doubling algorithm in a natural way by means of the decomposition of permutation $\Gamma_{(v)}$ in a perfect unshuffle which is internal ($[cycle, bus]$) and another which is external ($[PE, bus]$) to the PEs. In the particular case of one column with a single PE the

```
.......<-output-->...<-stage 6->...<-stage 5->...<-stage 4->...<-stage 3->...<-stage 2->...<-stage 1->
  cycle  3  2  1  0    3  2  1  0    3  2  1  0    3  2  1  0    3  2  1  0    3  2  1  0    3  2  1  0
PE bus
----------------------------------------------------------------------------------------------------------

0  0    48 32 16  0   24 16  8  0   12  8  4  0    6  4  2  0    3  2  1  0   33  1 32  0   48 32 16  0
   1    49 33 17  1   56 48 40 32   28 24 20 16   14 12 10  8    7  6  5  4   35  3 34  2   49 33 17  1

1  0    50 34 18  2   25 17  9  1   44 40 36 32   22 20 18 16   11 10  9  8   37  5 36  4   50 34 18  2
   1    51 35 19  3   57 49 41 33   60 56 52 48   30 28 26 24   15 14 13 12   39  7 38  6   51 35 19  3

2  0    52 36 20  4   26 18 10  2   13  9  5  1   38 36 34 32   19 18 17 16   41  9 40  8   52 36 20  4
   1    53 37 21  5   58 50 42 34   29 25 21 17   46 44 42 40   23 22 21 20   43 11 42 10   53 37 21  5

3  0    54 38 22  6   27 19 11  3   45 41 37 33   54 52 50 48   27 26 25 24   45 13 44 12   54 38 22  6
   1    55 39 23  7   59 51 43 35   61 57 53 49   62 60 58 56   31 30 29 28   47 15 46 14   55 39 23  7

4  0    56 40 24  8   28 20 12  4   14 10  6  2    7  5  3  1   35 34 33 32   49 17 48 16   56 40 24  8
   1    57 41 25  9   60 52 44 36   30 26 22 18   15 13 11  9   39 38 37 36   51 19 50 18   57 41 25  9

5  0    58 42 26 10   29 21 13  5   46 42 38 34   23 21 19 17   43 42 41 40   53 21 52 20   58 42 26 10
   1    59 43 27 11   61 53 45 37   62 58 54 50   31 29 27 25   47 46 45 44   55 23 54 22   59 43 27 11

6  0    60 44 28 12   30 22 14  6   15 11  7  3   39 37 35 33   51 50 49 48   57 25 56 24   60 44 28 12
   1    61 45 29 13   62 54 46 38   31 27 23 19   47 45 43 41   55 54 53 52   59 27 58 26   61 45 29 13

7  0    62 46 30 14   31 23 15  7   47 43 39 35   55 53 51 49   59 58 57 56   61 29 60 28   62 46 30 14
   1    63 47 31 15   63 55 47 39   63 59 55 51   63 61 59 57   63 62 61 60   63 31 62 30   63 47 31 15
```

Figure 6: Triplet $[cycle, PE, bus]$ for the example ($N = 64$, $Q = 8$, $r = 2$).

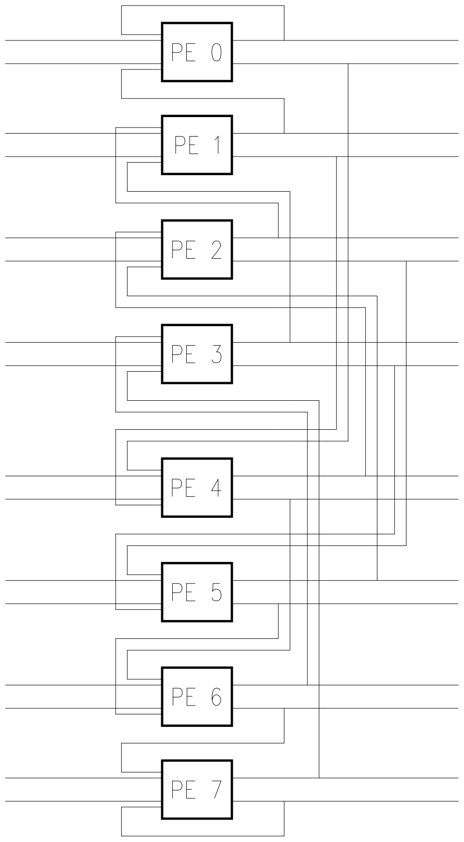

Figure 7: Perfect unshuffle network for the example of figure 4.

whole unshuffle will be internal, we obtain the solution of the previous section (sequence (20)), and the FIFO queues will have a length of $N - 2^v$ cells. In the extreme case of a column with r^{n-1} PEs the whole unshuffle will be external and, consequently, the FIFO queues will not be necessary.

The $[cycle, PE, bus]$ interpretation we have performed for the three dimensional representation $[x, y, z]$ of the index of each data item is not unique. Thus, for example, $[PE, cycle, bus]$ is another possible interpretation with similar characteristics, but with the order for carrying out the internal and external permutations changed. Consequently, we can establish the following result.

Theorem 2 *ACP successive doubling radix r and constant geometry algorithm of a sequence of N data items ($N = r^n$) can be carried out in a column of Q PEs ($Q = r^q$, $0 \leq q \leq n-1$) which implements (hardware translation) in each stage the following operator string*

$$B_{(v)}\Gamma_{(v)}^{PE,bus}\beta_{(v)}^{cycle,bus}\delta_{(v)}^{cycle,bus} \tag{30}$$

Where $v = \log_2 r$, the operators are applied from left to right, we consider the $[PE, cycle, bus]$ interpretation of the three dimensional representation of the index of each data item.

Its proof is similar to that of theorem 1.

The change in the order of operators Γ, β and δ in expressions (28) and (30) implies a different location for the PS and RS sections of the PE. Both designs share the same external interconnections (see figure 7, for example) and are identical when we have a column with a single PE. However, the initial distribution of the data (first stage) is different for each interpretation; with $[cycle, PE, bus]$ consecutive butterflies ($\{0,1\}$, $\{2,3\}$, ...) are processed in different PEs (see figure 6), whereas using $[PE, cycle, bus]$ they are processed in the same PE (a block of N/Q consecutive data items are processed in each PE).

V DCP SUCCESSIVE DOUBLING ALGORITHM

The constant geometry architecture of the DCP algorithm is obtained by performing a perfect shuffle of the data generated in each stage of the transform. Considering the three dimensional representation of the data indexes introduced in section IV (equation (21)) we define the order k perfect shuffle operator as

$$\sigma_{(k)}[x,y,z] = [[x_{u-k}\cdots x_1 y_w \cdots y_{w-k+1}],[y_{w-k}\cdots y_1 z_v \cdots z_{v-k+1}],[z_{v-k}\cdots y_1 x_u \cdots x_{u-k+1}]] \tag{31}$$

where $k \leq u + w + v$. We can easily generalize $\sigma_{(k)}$ in a similar way to equations (35) and (36). We are interested in the decomposition of permutation $\sigma_{(v)}$, $v =$

$\log_2 r$, into elementary permutations we can implement electronically. In order to achieve this we define two new operators: inverse decimation ($\bar{\delta}^{a,b}_{(k)}$) and inverse concatenation ($\bar{\beta}^{a,b}_{(k)}$), where $a, b = x, y, z$ ($a \neq b$)

$$\begin{aligned} \bar{\delta}^{x,z}_{(k)}[x,y,z] &= [[x_u \cdots x_1 z_k \cdots z_1], [y_w \cdots y_1], [z_v \cdots z_{k+1}]] & (32) \\ \bar{\beta}^{x,z}_{(k)}[x,y,z] &= [[x_{u-k} \cdots x_1], [y_w \cdots y_1], [z_v \cdots z_1 x_u \cdots x_{u-k+1}]] & (33) \end{aligned}$$

The electronic implementation of these two new operators is immediate. $\bar{\delta}_{(k)}$ can be realized by means of a multiplexor with 2^k inputs. $\bar{\beta}_{(k)}$ can be implemented by means of a FIFO queue of length N ($N = r^n$) with an input and 2^k outputs.

Lemma 3 *The following decompositions of the perfect shuffle permutation occur:*

$$\begin{aligned} \sigma^{x,z}_{(v)}[x,y,z] &= \bar{\delta}^{x,z}_{(v)} \bar{\beta}^{x,z}_{(v)}[x,y,z] & (34) \\ \sigma_{(v)}[x,y,z] &= \sigma^{y,z}_{(v)} \sigma^{x,z}_{(v)}[x,y,z] & (35) \end{aligned}$$

The proof is similar to those of lemmas 1 and 2 stated in previous sections.

Equations (35) and (36) guarantee the decomposition of the perfect shuffle permutation of the three dimensional representation of each data item's index. Consequently, we can establish the following result

Theorem 3 *DCP successive doubling radix r and constant geometry algorithm of a sequence of N data items ($N = r^n$) can be carried out in a column of Q PEs ($Q = r^q$, $0 \leq q < n$) which implement (hardware translation) each stage the following operator string*

$$\sigma^{PE,bus}_{(v)} \bar{\delta}^{cycle,bus}_{(v)} \bar{\beta}^{cycle,bus}_{(v)} B_{(v)} \tag{36}$$

Where $v = \log_2 r$, the operators are applied from left to right and we consider the $[cycle, PE, bus]$ interpretation of the three dimensional representation of the index of each data item.

Its proof is similar to that of theorem 1. Figure 8 shows the internal structure of the PE, where we have included two FIFO queues to facilitate the overlapping execution of different transform stages. This queues implement the operator string $\bar{\delta}^{cycle,bus}_{(v)} \bar{\beta}^{cycle,bus}_{(v)}$, whereas the partial perfect shuffle operator $\sigma^{PE,bus}_{(v)}$ determines the interconnection network of the PE column. Finally, we can establish in an immediate way the theorems for the DCP successive doubling algorithm which are equivalent to theorem 2 of section IV.

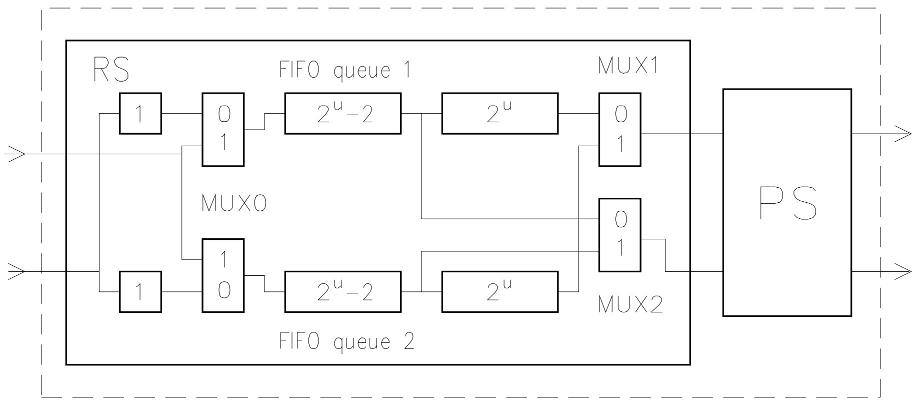

Figure 8: PE based on the operator string $\bar{\delta}_{(v)}\bar{\beta}_{(v)}B_{(v)}$ (algorithm DCP).

VI CONCLUSIONS

We can classify the hardware solutions for digital signal processing [17] into three groups: 1) conventional microprocessors; 2) specific domain microprocessors; and 3) application specific multiprocessors. The design based on conventional microprocessors is simple, but its greatest drawback is associated with their lack of specificity. The specific domain processors, among which we find the DSPs [1,48-50], are an attempt to optimize the design by limiting the application spectrum. They are specialized in algorithms whose fundamental operation is the summation of products. They are microprogrammed devices whose power is conditioned by the design of the databus. Moreover, hardware address generation is imperative to DSP performance since sample-by-sample data structure maintenance in software represents a substantial overhead [1]. An optimization of this second solution are the application specific DSPs , such as the FFT PDSP16510 processor from Plessey Semiconductor [3,37-38], among others [29,51].

Parallelism has long been considered an important solution to DSP applications requiring higher computational power than a single processor could offer. A first approach consists in designing DSPs with multiple arithmetic units and sophisticated addressing modes for accessing the data located in multiple memory banks [1,37-38]. However, the solution is in the design of multiprocessor architectures which permit the exploitation of the inherent parallelism of DSP algorithms. In order to do this we must consider two important problems this new alternative presents: To carry out the partitioning of the algorithm and to choose the type of synchronization between PEs, which must also be solved in an effective way. Partitioning is very important to the DSP system design process [1]. Current reality is that multiple DSP chip architectures are limited to two or three devices with considerable restrictions on interprocessor communication and consequently on problem partitioning. In this work we have shown that it is possible to design an application specific DSP multiprocessor architecture which can be integrated in VLSI or WSI technology.

The successive doubling method is an efficient procedure for the design of fast algorithms for orthogonal transforms. Algorithms which present a high spatial parallelism in the calculation stages and an inherent sequentiality between them. Consequently, the appropriate architecture will be a rectangular array made up of a pipeline of PE columns. The application specific multiprocessor we have proposed in this work efficiently solves the problems mentioned before. The result is a constant geometry systolic architecture. The geometry is determined by the perfect unshuffle (or perfect shuffle) permutation of the data generated in each stage of the ACP (or DCP) algorithm.

The constant character of the geometry permits the implementation of the data flow of the successive doubling algorithm by means of hardwired control. That is, the PEs do not need address arithmetic units to locate the data. Addressing is inherent to the evolution of the data in the FIFO queues of the RS section and the external interconnection network. Moreover, the partitioning of the algorithm appears in a natural way when this permutation (perfect unshuffle or shuffle) is decomposed into a string of elementary permutations which can be implemented electronically: decimation, concatenation and partial perfect unshuffle (ACP) or shuffle (DCP). Finally, we have chosen the systolic operating mode, which is an effective synchronization method.

The design of the application specific architecture for each one of the transforms is completed with the design of the PS section. In the companion paper [93] we will give a unified presentation of the PS section for the different transforms we consider. More specifically, we will use a CORDIC arithmetic unit, as it has a better throughput per unit area than VLSI multipliers.

References

[1] H.M. Ahmed, "Directions in DSP processors". IEEE J. on Selected Areas in Communications, Vol. 8, No. 8, pp. 1420-1427, 1990.

[2] N.T. Ahmed, T. Natarajan and K.R. Rao. "Discrete Cosine Transform". IEEE Trans. on Computers, Vol. C-23, No. 1, pp. 90-93, 1974.

[3] R.D. Albon, G.E. Floyd y J.E. Coles. "A Mask Programmable DSP Array". IEEE 1989 Custom Integrated Circuits Conference (IEEE Press, New York), pp. 2011-2013, 1989.

[4] H.M. Alnuweiri, "A new class of optimal VLSI networks for multidimensional transforms". 1990 Int'l Conf. on Parallel Processing (IEEE Catalog), Vol. I, pp. 449-456, 1990.

[5] A. Antola, R. Negrini, M.G. Sami and N. Scarabottolo, "Policies for fault-tolerance through mixed space- and time-redundancy in semi-systolic FFT

arrays". Int'l Conf. Systolic Arrays (IEEE Catalog), San Diego, CA, pp. 565-576, 1988.

[6] A. Averbuch, E. Gabber, B. Gordissky and Y. Medan, "A parallel FFT on an MIMD machine". Parallel Computing, Vol. 15, pp. 61-74, 1990.

[7] G.D. Bergland, "A fast Fourier transform algorithm for real-valued series". Commun. ACM, Vol. 11, No. 10, pp. 703-710, 1968.

[8] G. Bi and E.V. Jones, "A pipelined FFT processor for word-sequential data". IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-37, No. 12, pp. 1982-1985, 1989.

[9] G. Bilardi and M. Sarrafzadeh, "Optimal discrete Fourier transform in VLSI". In VLSI: Algorithms and Architectures, P. Bertolazzi and F. Luccio (Editors) (North-Holland, Amsterdam), pp. 79-89, 1985.

[10] G. Bilardi and M. Sarrafzadeh, "Optimal VLSI circuits for the discrete Fourier transform". In Advances in Computing Research, F.P. Preparata (Ed.) (JAI Press, London), Vol. 4, pp. 87-101, 1987.

[11] R.E. Blahut, "Fast algorithms for digital signal processing". (Addison-Wesley, Reading, MA), 1985.

[12] G. Bongiovanni, "Two VLSI structures for the discrete Fourier transform". IEEE Trans. on Computers, Vol. C-32, No. 8, pp. 750-754, 1983.

[13] G. Bongiovanni, "A VLSI network for variable size FFT's". IEEE Trans. on Computers, Vol. C-32, No. 8, pp. 756-760, 1983.

[14] R.N. Bracewell, "Fast Hartley transform". (Oxford University Press, New York), 1986.

[15] E.O. Brigham, "The fast Fourier transform". (Prentice Hall, Englewood Cliffs, NJ), 1974.

[16] C.S. Burrus, "Index mappings for multidimensional formulation of the DFT and convolution". IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-25, No. 3, pp. 239-242, 1977.

[17] F. Catthoor and H.J. de Man, "Application-specific architectural methodologies for high throughput digital signal and image processing". IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-38, No. 2, pp. 339-349, 1990.

[18] P. Chow, Z.G. Vranesic and J.L. Yen, "A pipeline distributed arithmetic PFFT processor". IEEE Trans. on Computers, Vol. C-32, No. 12, pp. 1128-1136, 1983.

[19] J.W. Cooley and J.W. Tukey, “An algorithm for the machine calculation of complex Fourier series”. Math. Comput., Vol. 19, No. 4, pp. 297-301, 1965.

[20] G.L. Demuth, “Algorithms for defining mixed radix FFT flow graphs”. IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-37, No. 9, pp. 1349-1358, 1989.

[21] P. Duhamel and H. Hollmann, “Split-radix FFT algorithm”. Electronics Letters, Vol. 20, No. 1, pp. 14-16, 1984.

[22] R.D. Fellman. “Design Issues and a Architecture for the Monolitic Implementation of a parallel Digital Signal Processor”. IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-38, No. 5, pp. 839-852, 1990.

[23] R.D. Fellman, R.T. Kaneshiro y K. Konstantinides. “Design and Evaluation of an Architecture for a Digital Signal Processor for Instrumentation Applications”. IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-38, No. 3, pp. 537-546, 1990.

[24] J.A.B. Fortes, K. Fu y B.W Wah. “Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays”. En Computer Architecture (V. Milutinovic, Ed.) (North Holland, New York), pp. 454.494, 1988.

[25] P.D. Gader, “Bidiagonal factorization of Fourier matrices and systolic algorithms for computing discrete Fourier transforms”. IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-37, No. 8, pp. 1280-1283, 1989.

[26] I. Gertner and M. Shamash, “VLSI architectures for multidimensional Fourier transform processing”. IEEE Trans. on Computers, Vol. C-36, No. 11, pp. 1265-1274, 1987.

[27] I. Gertner, “A new efficient algorithm to compute the two-dimensional discrete Fourier transform”. IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-36, No. 7, pp. 1036-1050, 1988.

[28] C. Gimarc, V. Milutinovic and O. Ersoy, “Time complexity modeling and comparison of parallel architectures for Fourier transform oriented algorithms”. 22th Hawaii Int’l Conf. Vol I: Architecture (IEEE Catalog), pp. 160-170, 1989.

[29] S. Gomez, S. Gonzalez, D.D. Hsu and A.E. Kuo, “An application-specific FFT processor”. Electronic Engineering, Vol. 60, No. 783, pp. 99-106, 1988.

[30] R.C. González and P. Wintz, “Digital image processing”. (Addison-Wesley, London), 1977.

[31] S. Gudvangen and A.G.J. Holt, "Computation of prime factor DFT and DHT/DCCT algorithms using cyclic and skew-cyclic bit-serial semisystolic IC convolvers". IEE Proceedings, Part G, Vol. 137, No. 5, pp. 373-389, 1990.

[32] R.V.L. Hartley, "A more symmetrical Fourier analysis applied to transmission problems". Proc. of the IRE, Vol. 30, pp. 142-150, 1942.

[33] D.T. Harper III, "Block, multistride vector, and FFT accesses in parallel memory systems". IEEE Trans. on Parallel and Distributed Systems, Vol. PDS-2, No. 1, pp. 43-51, 1991.

[34] M.T. Heideman and C.S. Burrus, "On the number of multiplications necessary to compute a length-2^n DFT". IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-34, No. 1, pp. 91-95, 1986.

[35] P. Hertz, "An algorithm for the fast Fourier transform on a Connection Machine". J. Computers in Physics, Vol. 4, No. 1, pp. 86-90, 1990.

[36] H. Hikawa and V.K. Jain, "20 Million samples/s wafer processor FFT architecture". In Signal Processing V: Theories and Applications, L. Torres, E. Masgrau and M.A. Lagunas (eds.) (Elsevier Science Publ., Amsterdam), Vol. 1, pp. 9-16, 1990.

[37] B. Holland and J. Mather, "Monolithic frequency domain processing with 450 MFLOPS throughput". Electronic Engineering, Vol. 61, No. 752, pp. 29-36, 1989.

[38] B. Holland, J. Mather and S. Brightfield, "Fourier chip tackles transforms". ESD: The Electronic System Design Magazine, pp. 30-36, July, 1989.

[39] S. Horiguchi and T. Nakada, "Experimental performance evaluation of parallel fast Fourier transform on a multiprocessor workstation". 1990 Int'l Conf. on Parallel Processing (IEEE Catalog), Vol. III, pp. 97-101, 1990.

[40] IEEE Acoustics, Speech, and Signal Processing Society, "Programs for digital signal processing". (IEEE Catalog), 1979.

[41] J. Jaja and R.M. Owens, "Optimal algorithms for mesh-connected parallel processors with serial memories". In Advances in Computing Research, F.P. Preparata (Ed) (JAI Press, London), Vol. 4, pp. 103-115, 1987.

[42] L.H. Jamielson, P.T. Mueller Jr and H.J. Siegel, "FFT algorithms for SIMD parallel processing systems" J. Parallel and Distributed Computing, Vol. 3, No. 1, pp. 48-71, 1986.

[43] C.R. Jesshope, "The implementation of fast radix 2 transforms on array processors", IEEE Trans. on Computers, Vol. C-29, No. 1, pp. 20-27, 1980.

[44] L. Johnsson, C. Ho, M. Jacquemin and A. Ruttenberg, "Systolic FFT algorithms on boolean cube networks". Int'l Conf. on Systolic Arrays (IEEE Catalog), San Diego, CA, pp. 151-161, 1988.

[45] J. Jou and J.A. Abraham, "Fault-tolerant FFT networks". IEEE Trans. on Computers, Vol. C-37, No. 5, pp. 548-561, 1988.

[46] M. Kosaka and Z. Segall, "Performance considerations for parallel FFT algorithms". 23th Annual Hawaii Int'l Conf. on System Sciences (IEEE Catalog), Vol. I. pp. 261-267, 1990.

[47] H.T. Kung and C.E. Leiserson, "Systolic arrays for VLSI", in Sparse Matrix Proc. 1978, I.S. Duff and G.W Steward, (Editors) (Soc. Indust. Appli. Math.), pp. 256-282, 1979.

[48] E.A. Lee. "Programmable DSP Architectures: Part I". IEEE ASSP Magazine, Vol. 5, No. 4, pp. 4-19, 1988.

[49] E.A. Lee. "Programmable DSP Architectures: Part II". IEEE ASSP Magazine, Vol. 6, No. 1, pp. 4-14, 1989.

[50] E.A. Lee. "Programmable DSPs: A brief overview". IEEE Micro, Vol. 10, No. 5, pp. 14-16, 1990.

[51] G. Lulkuo, M. Fleming and S. Magar, "A 500 MOPS DSP chip set". Electronic Engineering, Vol. 60, No. 738, pp. 109-113, 1988.

[52] R.N. Mahapatra, V. Ashok-Kumar, B.K. Das and B.N.Chatterji, "Performance of parallel FFT algorithm on multiprocessor". 1990 Int'l Conf. on Parallel Processing (IEEE Catalog), Vol. III, pp. 368-369, 1990.

[53] W. Marwood and A.P. Clarke, "Matrix product machine and the Fourier transform". IEE Proceedings, Part G, Vol. 134, No. 4, pp. 295-301, 1990.

[54] B.C. McKinney and F. El-Guibaly, "VLSI design of an FFT processor network". J. Integration, Vol. 8, No. 3, pp. 301-320, 1989.

[55] D.I. Moldovan y J.A.B. Fortes. "Partitioning and Mapping Algorithms into Fixed Size Systolic Arrays". IEEE Trans. on Comput., Vol. C-35, No. 1, pp. 1-12, 1986.

[56] J.H. Moreno y T. Lang. "Matrix Computations on Systolic Type Meshes: An Introduction to the Multimesh Graph Method". IEEE Computer, No. 4, pp. 32-51, 1990.

[57] J.J. Navarro, J.M. Llabería y M. Valero. "Partitioning: An Essential Step in Mapping Algorithms into Systolic Array Processors". IEEE Computer, Vol. 20, No. 7, pp. 77-89, 1987.

[58] A. Norton and A.J. Silberger, "Parallelization and performance analysis of the Cooley-Tukey FFT algorithm for shared memory architectures", IEEE Trans. on Computers, Vol. C-36, No. 5, pp. 581-591, 1987.

[59] H.J. Nussbaumer, "Fast Fourier transform and convolution algorithms". (Springer-Verlag, Berlin), 1981.

[60] D. Parker, "Notes on shuffle/exchange-type switching networks". IEEE Trans. on Computers, Vol. C-29, No. 3, pp. 213-222, 1980.

[61] M.C. Pease, "An adaptation of the fast Fourier transform for parallel processing". J. Ass. Comput. Mach., Vol. 15, pp. 252-264, 1968.

[62] M.C. Pease, "The indirect binary n-cube microprocessor array", IEEE Trans. on Computer, Vol. C-26, No. 5, pp. 458-473, 1977.

[63] F.P. Preparata and J. Vuillemin, "The cube connected cycles: A versatil network for parallel computation", Communications of the ACM, Vol. 25, No. 5, pp. 300-309, 1981.

[64] L.R. Rabiner and B. Gold, "Theory and application of digital signal processing". (Prentice Hall, Englewood Cliffs, NJ), 1975.

[65] L.R. Rabiner and C.M. Rader, (Eds.), "Digital Signal Processing". (IEEE Press, New York), 1972.

[66] M.A. Richards, "On hardware implementation of split-radix FFT". IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-36, No. 10, pp. 1575-1581, 1988.

[67] P.R. Roeser and M.E. Jernigan, "Fast Haar transform algorithms". IEEE Trans. on Computers, Vol. C-31, No. 2, pp. 175-177, 1982.

[68] M.R. Samatham and D.K. Pradham, "A multiprocessor network suitable for single chip VLSI implementation". 11th Int'l Symp. on Comput. Architec. (IEEE Catalog), pp. 328-337, 1984.

[69] G. Sande, "Fast Fourier transform - A globally complex algorithm with locally real implementations". 4th Annu. Princeton Conf. Inform. Sci. Syst. (Princeton Univ., Princeton, NJ), pp. 136-142, 1970.

[70] K. Sapiecha and R. Jarocki, "Modular architecture for high performance implementation of the FFT algorithm". IEEE Trans. on Computers, Vol. C-39, No. 12, pp. 1464-1468, 1990.

[71] W. Shen and A.Y. Oruc, "Systolic arrays for multidimensional discrete transforms". J. Supercomputing, Vol. 4, No. 3, pp. 201-222, 1990

[72] S.G. Smith, “Fast Fourier Machines”. In VLSI Signal processing: A bit-serial approach, P. Denyer and D. Renshaw (Editors). (Addison-Wesley, Wokingham, UK), pp. 147-199, 1985.

[73] Y. Solowiejczyk and J. Petzinger, “The radix 4 FFT on a multiprocessor shared memory system”. 1990 Int'l Conf. on Parallel Processing (IEEE Catalog), Vol. III, pp. 362-363, 1990.

[74] H. Sorensen, M.T. Heideman and C.S. Burrus, “On computing the split-radix FFT”. IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-34, No. 1, pp. 152-156, 1986.

[75] H.S. Stone, “Parallel Processing with the perfect shuffle”. IEEE Trans. on Computers, Vol. C-20, No. 2, pp. 153-161, 1972.

[76] E.E. Swartzlander, Jr, “VLSI signal processing systems”, (Kluwer Academic Publ., Boston), 1986.

[77] D. Tao, C.R.P. Hartmann and Y.S. Chu, “A novel concurrent error detection scheme for FFT networks”. 20th Int'l Symp. on Fault-Tolerant Computing (IEEE Catalog), pp. 114-121, 1990.

[78] C.D. Thompson, “Fourier Transforms in VLSI”. IEEE Trans. on Computers, Vol. C-32, No. 11, pp. 1047-1057, 1983.

[79] P. Tortoli and F. Andreuccetti, “A high-speed FFT unit based on a low cost digital signal processor”. IEEE Trans. on Circuits and Systems, Vol. CS-35, No. 11, pp. 1434-1438, 1988.

[80] T.K. Truong, I.S. Reed, I. Hsu, H. Shyu and H.M. Shao, “A pipeline design of a fast prime factor DFT on finite field”. IEEE Trans. on Computers, Vol. C-37, No. 3, pp. 266-273, 1988.

[81] D. Tsai and M. Vulis, “Computing discrete Fourier transform on a rectangular data array”. IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-38, No. 2, pp. 271-276, 1990.

[82] M. Vetterli and H.J. Nussbaumer, “Simple FFT and DCT algorithms with reduced number of operations”. Signal Processing, Vol. 6, No. 8, pp. 267-278, 1984.

[83] W. Wlde and O. Haan, “Performance of fast Fourier transforms on vector-computers”. J. Supercomputer, 40, pp.42-49, 1990.

[84] S.A. White, “A very high speed FFT architecture”. 22th Asilomar Conf. on Signal Systems and Computers (Maple Press), Vol. I, pp. 352-357, 1989.

[85] S. Winograd, "On computing the discrete Fourier transform". Math. Comput., Vol. 32, No. 3, pp. 175-199, 1978.

[86] E. Wold and A.M. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementations". IEEE Trans. on Computers, Vol. C-33, No. 5, pp. 414-426, 1984.

[87] K.L. Wong and W.C. Siu, "A new systolic structure for the realization of recursive discrete Fourier transform". In Performance of Distributed and Parallel Systems (T. Hasegawa, H. Takagi and Y. Takahashi, Eds.) (North Holland), pp. 439-454, 1989.

[88] H.R. Wu and F.J. Paoloni, "On the two-dimensional vector split-radix FFT algorithm". IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-37, No. 8, pp. 1302-1304, 1989.

[89] K. Yamashita, A. Kanasugi, S. Hijita, G. Goto, N. Matsumura and T. Shirato, "A wafer-scale 170000-gate FFT processor with built-in circuits". IEEE J. Solid State Circuits, Vol. SC-23, No. 2, pp. 336-342, 1988.

[90] K. Yamashita, A. Kanasugi, S. Hijita and G. Goto, "A wafer-scale FFT processor featuring a repeatable building block". Int'l Conf. on Wafer Scale Integration (IEEE Catalog), San Francisco, CA, pp. 299-307, 1989.

[91] J. You and S.S. Wong, "A high performance single chip FFT array processor for WSI". Int't Conf. on Wafer Scale Integration (IEEE Catalog), pp. 60-67, 1990.

[92] E.L. Zapata, F.F. Rivera, I. Benavides, J.M. Carazo and R. Peskin, "Multidimensional fast Fourier transform into SIMD hypercubes". IEE Proceedings Part E: Computers and Digital Techniques, Vol. 137, No. 4, pp. 253-260, 1990.

[93] F. Argüello, "Application-specific architecture for fast transforms based on the successive doubling method, Part II: Orthogonal transforms".