Print ISSN:1312 – 2622; Online ISSN: 2367-5357
DOI:10.2478/itc-2013-0010

Investigation of the Indirect Hypercube as Natural Architecture for Parallel Algorithms of a Transpose Type for FFT and other Fourier-Related Transforms

Ph. Philipov, V. Lazarov

Key Words: *Parallel FFT; parallel FHT; parallel real-valued FFT; parallel FCT; indirect hypercube; high-performance computer architectures.*

Abstract. *The natural architectures are architectures, derived from the signal graph of the corresponding algorithm. That is why they are considered to be the most appropriate architectures for parallel realization of this algorithm. For Fast Fourier Transform algorithm (FFT) two types of natural architectures are known – the direct and the indirect hypercube. The direct hypercube has been investigated and analyzed a long time ago. The development of the concept of Indirect Hypercube, although quite old, is too difficult, controversal and still unfinished. Fast Hartley transform (FHT)/Real-valued Fast Fourier transform (RFFT) algorithms are important Fourier-related transforms, because they lower twice the operational and memory requirements when the input data is real-valued. These types of algorithms, however, have an irregular computational structure, which makes their parallel implementation a very difficult task. The aim of this paper is, based on the results achieved so far, to present further development of the concept Indirect Hypercube. A method of parametric synthesis of an indirect hypercube is described as a model of parallel FFT algorithms of a transpose type with different granularity/radix. This method is generalized for relevant RFFT/FHT and FCT algorithms. Two types of SIMD array architectures are described (radix-2 and radix-4), based on the indirect hypercube concept. These architectures are implemented as fast FFT/RFFT/FHT processors for real time applications. The performance estimation, as well as the estimation of resource utilization is carried out.*

Introduction

Fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. The discovery of this algorithm by Cooley and Tukey [Cooley-1] in 1965 is a key point in the development of digital algorithms and parallel calculations. FFT plays a significant role in several important scientific and technical areas, such as solving partial differential equations, spectral analysis, digital signal processing, image processing, etc. Hence, its impact over the development of high-performance architectures and digital algorithms is large.

In many cases, from a performance viewpoint, parallel implementation of these algorithms is necessary.

It is known that parallelism is an inherent feature of FFT. The development of parallel FFT algorithms and their implementation on different computing platforms began in the early 70-ies of the XX-th century.

Parallel implementations of FFT have been made on almost all high-performance parallel computer architectures – vector computers, systolic arrays, transputer systems, superscalar and VLIW processor architectures, processor networks with different organization, etc. Considerable efforts have been made in studies for the realization of FFT on massive parallel architectures of processors, starting with synchronous SIMD approaches, which later evolved to MIMD asynchronous approaches.

Leading factors in this process are efficiency, performance, latency and scalability.

Nowadays, the modern research in the field of parallel FFT algorithms and architectures is concentrated in two main trends.

The first trend, that can be conditionally called a network trend, is developed for the goals and tasks of computational mathematics, e.g. *Fast Poisson Solvers*, based on FFT are considered to be among the fastest techniques for solving the Poisson equation on uniform grids (grids). This trend is associated with the development of the so called *massive parallelism* and modern MIMD computer architectures. It is based on multiple processors working in network (clusters), distributing data between processors in accordance with some parallel algorithm (e.g. 2D Transpose algorithm), and exchanging data in a standard way (e.g. MPI-message passing interface, active packets, etc.). Essential for this trend is the efficiency of the pair algorithm-architecture when the problem size and the number of processors increase significantly.

The second trend, which can be conditionally called *single-chip* trend, includes efforts to develop single-chip fast FFT processors operating in real time which are necessary for the needs of many applications in communications (high speed OFDM-based communication technologies, such as ADSL, VDSL, DVB, FFT-based 2D image processing, etc., requiring high-speed FFT processors operating in real time.

In many applications the input data for the DFT are real-valued and the outputs exhibit conjugational symmetry, or the inputs exhibit conjugational symmetry and in this case the outputs are real values.

The discrete Hartley transform (DHT) [Bracewell-1] is a Fourier-related transform of discrete, periodic data, similar to DFT with analogous applications in signal processing and related fields. Its main distinction from DFT is that it transforms real data, thus being an alternative for the real-valued DFT algorithms.

Specialized real-valued FFT algorithms (RFFT) [Sorensen-1] and fast Hartley transform (FHT) [Bracewell-2] have been designed for this situation in order to remove the redundant operations, lowering twice the memory and operational requirements.

FHT/RFFT algorithms are important, because with their help other *Fourier-related* transforms can be obtained, e.g. discrete cosine transforms (DCT), discrete sine transforms (DST), etc.

Parallel implementations of FHT/RFFT have been made for some types of parallel architectures. Nevertheless, these algorithms have an irregular computational structure which makes their efficient parallelization a very difficult task.

For nearly 50-years period a huge amount of literature has been written on FFT. There are authors who consider this subject exhausted (especially for the complex FFT). Still, there are some problems that are not yet sufficiently studied and deserve exploring. Such a problem is the problem connected with the natural architectures of these algorithms.

Natural architectures for a given algorithm are such architectures, which are derived directly from the data flow, (signal graph) of this algorithm. For this reason, they are considered to be the most appropriate architectures for parallel realization of this algorithm. For FFT, two types of natural architectures are known – the *direct* and *indirect* hypercube.

The Direct Hypercube has been studied in details and there is hardly anything else to add on the subject.

Stone [Stone-1] and Pease [Pease-1] were the first to highlight the potential of the Indirect Hypercube.

In 1995 Macceo et al. [Macceo-1], on the base of the butterfly diagram, map the algorithm on an architecture of an indirect binary Hypercube type. It is a conveyor type architecture and the workload of the first stage of processors is significantly higher than that of the other stages. This circumstance leads to decrease of the efficiency.

Gradually over time the characteristics of the indirect Hypercube seem to pass on to direct Hypercube. Since then one can see in the references a strange mixture of both types of the Hypercube. Gradually, the prevailing opinion is that the direct Hypercube is the only natural architecture for FFT algorithms.

In 2005 an enigmatic array architecture has appeared [Zhenyu Liu-1]. This architecture offers the following advantages:

• Eliminating the transfer of data between stages, resulting in both improved latency and system performance.

• Elimination of the conflicts in memory.

• System performance increases linearly with increasing the number of processors.

Although the authors do not use *indirect hypercube* for characterization of this architecture, its internal features coincide with the respective features of the indirect hypercube. That is why we consider this architecture to be a hidden version of the indirect hypercube.

The purpose of this paper is the further development and reviewing of the potential of the concept Indirect Hypercube, including:

• Giving a clear form and outlining the intrinsic properties of this architecture.

• Generalization of the concept with respect to granularity/radices.

• Generalization of the concept with respect to other Fourier-related transforms (real-valued FFT, FHT, FCT).

• Some applications.

FFT Analysis

DFT

Discrete Fourier Transform (direct and inverse) is defined as:

$$(1.a)\quad F(\mathrm{k}) = \sum_{\mathrm{n}=0}^{\mathrm{N}-1} X(\mathrm{n})\,\mathrm{W}^{-\mathrm{kn}}$$

$$(1.b)\quad X(\mathrm{n}) = \frac{1}{\mathrm{N}} \sum_{\mathrm{k}=0}^{\mathrm{N}-1} F(\mathrm{k})\,\mathrm{W}^{k\mathrm{n}}$$

$$W = e^{i2\pi/N} = \cos(2\pi/N) + i\sin(2\pi/N).$$

FFT Description

Radix-2 Fast Fourier Transform (Cooley-Tukey) is an algorithm, realizing DFT. The algorithm is very suitable for $N = 2^n$. It is based on the so called *Danielson-Lanczos lemma* – formation of two N-point sequences ($X_{f(k)}$ and $X_{f(N/2+k)}$) on the base of two (even and odd) $N/2$-point sequences ($X_{f0(k)}$ and $X_{f1(k)}$). There are two basic versions (schemes) of this algorithm – DIT and DIF. The formation of the N-point sequences is in accordance with the following relations (simplified butterflies) [Singleton-1]:

DIT

$$(2.a)\quad \begin{aligned} X_{f(k)} &= X_{f0(k)} + e^{-i(2\pi/N)k}.X_{f1(k)} \\ X_{f(N/2+k)} &= X_{f0(k)} - e^{-i(2\pi/N)k}.X_{f1(k)}, \quad 0 \le k \le N/2 - 1. \end{aligned}$$

DIF

$$(2.b)\quad \begin{aligned} X_{f(k)} &= X_{f0(k)} + X_{f1(k)} \\ X_{f(N/2+k)} &= (X_{f0(k)} - X_{f1(k)})\, e^{-i(2\pi/N)k}, \quad 0 \le k \le N/2 - 1. \end{aligned}$$

Let $N = 2^n$.

The essence of both schemes consists in the consecutive calculation of n groups of intermediate weighted sums (butterflies) in accordance with (2.a)/(2.b).

If we analyze the input and output indices (as binary numbers) of these sums, we see that:

1)The input indices in a given sum differ in their most significant digit (all other digits are equal).

2) The output indices can be obtained through left cyclic rotation of the respective input indices; 3) The indices of the final sums are obtained in a bit reversed order.

Items 1) and 2) specify the so called *perfect shuffle* (PS) method of mixing an array. That is why if we want to present the flow of data in this algorithm, the best way is to use any PS model.

Perfect Shuffle

Mixing of an array N ($N = 2^k$) according to the *perfect shuffle* method mathematically is described as follows:

$$PS(n) = 2 * n \quad \text{for } n < N/2 \qquad (3)$$
$$PS(n) = 2 * n - N + 1 \quad \text{for } n \geq N/2$$

If the indices are presented as binary numbers, it is not difficult to see that:

• The objects of shuffling form pairs.

• The indices of the input pairs differ in their most significant digit (all other digits are equal).

• The output indices (after shuffling) can be obtained by means of left cyclic rotation of the respective input indices.

Perfect Shuffle Models

Omega network and Butterfly network are interconnection networks which are used in multiprocessor computer architectures. They are multistage interconnection networks [Bhuyan-1] whose basic property is the perfect shuffling of the input signals. They contain switching elements (crossbar switches) which are arranged in columns (stages).

Though Omega network and Butterfly network are topologically equivalent (the difference between them is in the manner of arrangement of the switches), they underlie two basic approaches for machine realization of PS method.

Conceptually, two approaches are possible to perform a perfect shuffle on an array.

First approach. In shuffling each element retains its position (the memory address where it is stored) and changes its index. An illustration of this approach is the well-known butterfly diagram (constructed on the base of the butterfly network), so we can conditionally call it the *Butterfly approach*.

Second approach. In shuffling each element changes its position in correspondence with its new index. An illustration of this approach is the Omega network, so that we can conditionally call it the *Omega approach*.

Generally the Butterfly approach does not require data buffering while Omega approach requires buffering of data. Perhaps for this reason, conventional (uniprocessor, sequential) FFT implementations are usually based on this approach, and the butterfly diagram has established itself as an essential tool for presenting FFT flow of data (primarily for conventional serial algorithms, subsequently for parallel algorithms).

When speaking about parallel multiprocessor implementation, we observe two levels of perfect shuffling:

• Low level (intra-processor level) concerning operations (perfect shuffling) inside given processor.

• High level (inter-processor level) concerning interprocessor communications.

Depending on the high-level PS approach implementation there exist generally two types of parallel algorithms:

• Data exchange type algorithms (butterfly approach) (e.g. binary exchange algorithm). These algorithms have the direct binary hypercube as natural architecture (directly map on).

• Transpose type algorithms (Omega approach) (e.g. 2D transpose algorithm). These algorithms have as natural architecture (directly map on) different types of an indirect binary hypercube.

Omega Network

The main characteristics of an Omega network are:

• It has $2^n = N$ inputs and so many outputs.

• It is composed of $\log_2 N$ stages, each stage including $N/2$ switching elements (crossbar switches).

• Every crossbar switch has two inputs (0 and 1) and two outputs (0 and 1).

Technically, the interconnection between the switching elements of successive stages is accomplished in correspondence with the following procedure:

• Every switch receives a number (binary) within a given stage.

• A personal identifier is associated with every input/output of a given switch.

• The input identifiers include the input number (0 or 1) as the most significant digit, followed by the switch number.

• The output identifiers include the switch number as the most significant part, followed by the output number (0 or 1).

• The output identifiers of a given switch can be obtained by means of left cyclic rotation of the respective input identifiers.

• The interconnection between consecutive stages is accomplished on the basis of the coincidence of input and output identifiers.

FFT Flow of Data

Omega network is a very suitable model for viewing the flow of data of radix-2 DIT (or DIF) FFT. Each stage of this network is associated with the generation of a given group of sums. Every switch of a given stage obtains two partial sums from the previous stage and sends two sums to the next stage. When the input data is in a natural order, the final results are in a bit-reversed order. And vice versa – when the input data is in a bit-reversed order, the final results are in a natural order. *Figure 1* shows well the FFT parallel intrinsic features – the parallel threads, as well as the homogeneity of the flow of data from a stage to a stage. This homogeneity makes possible the operation of all stages of the FFT algorithm to be performed by one physical stage. This is accomplished when the outputs of one stage of the Omega network are fed to the inputs of the same stage in accordance with the presented manner of interconnection (*figure 2*). The structure, presented in *figure 2* is known as one-stage indirect binary hypercube, and it can be regarded as the maximum parallel structure of 8-point FFT of this type (8-point FFT on 4 processors).

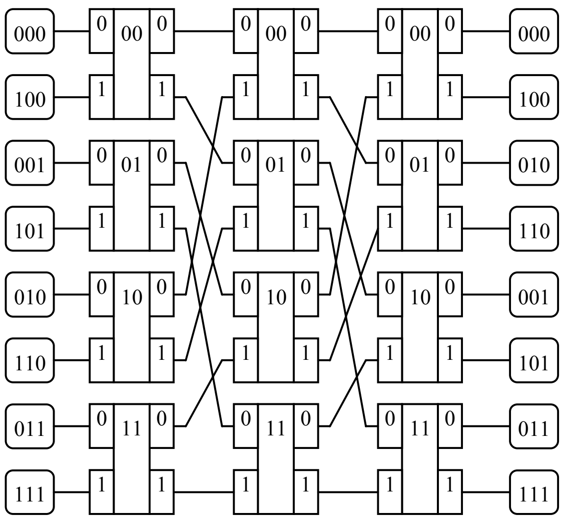

Figure 1. 8x8 Omega network hypercube

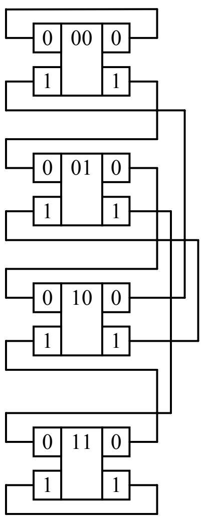

Figure 2. 8x8 1-stage indirect

Parametrical Synthesis of the Indirect Hypercube

It is clear that the *maximum parallel structure* is not the best solution, and that it is preferable (recommendable) that we could use efficiently the structure from *figure 2* (including 4 PE and very well suited for 8-point FFT) for FFT of greater number of points – 16, 32, etc. In other words, it is necessary to have a method for parametrical synthesis of such structures, where the parameters are the number of points and the number of processors.

Let $N = 2^n$, $n = n_1 + n_2$, $0 < n_1 < n$. We intend to construct an analogous parallel structure for N-point FFT on the basis of 2^{n1} *generalized crossbar switches*.

The generalized crossbar switch (GCS) of 2´2 dimension (two inputs and two outputs) is an integrated unified object, including a set of ordinary 2´2 crossbar switches whose n_1 least significant binary digits of their numbers coincide (GCS number). GCS is considered as a complex structure which includes one simple 2´2 crossbar switch, performing the functions of a processor element, and one dual-port memory (DM) which serves for reading of the input data for PE and storing the intermediate or final results.

Two types of identifiers are extracted from the global identifier (consisting of n binary digits) – a *Group identifier* and a *Local identifier*.

The Group Identifier (GI) includes the n_1 + 1 least significant digits of the global identifier.

• GI identifies the groups of signals entering/exiting a given GCS input/output and is of two types – an input GI and an output GI.

• The input GI includes the GCS input number as the most significant digit (0 or 1), followed by the GCS number.

• The output GI includes the GCS number as the most significant part, followed by the GCS output number (0 or 1).

• The output GI of a given output is obtained through left cyclic rotation of the corresponding input GI.

• GCS interconnection is based on coincidence of the input and output GI.

The Local Identifier (LI) includes the n_2 most significant digits of the global identifier.

• LI identifies the signals inside GCS.

• The most significant digit of LI indicates the PE input number.

• The least significant digit of LI indicates the GCS input number.

Basic property of LI – left cyclic rotation at transitions from a stage to a stage.

Generalization of the Concept for an Indirect Hypercube with Respect to Granularity

Transpose type Algorithms

If we try to define the algorithm described in the previous item, we can classify it as an algorithm of transpose type with the finest granularity (two). In general, this type of algorithms can be described in the following way.

Consider a q-dimensional transpose algorithm on $P = 2^p$ processors. In this case $N = G^q$, where G ($G = 2^g$) is granularity. This algorithm must contain q operational phases and between each phase transposition of the q-dimensional array is performed. During the operational phase each processor performs FFT in a particular direction (current rows of the array). Every transposition represents a left g-digit cyclic rotation of the indices. The number of processors, k, to which a given processor communicates, is determined by the expression $k = \min(P, G)$. The set of processors with which a given processor communicates does not change at different transposition phases.

Perfect Shuffle Sequences

It was already shown that the main feature, on which the use of the Omega network is based, is the ability to perform (and to be a suitable model) of a perfect shuffle. Constructing such a model for transpose algorithms with higher granularity presupposes presentation as a single act (in one stage) the sequence of shuffles (e.g. a sequence of 2 *PS* in the case of an algorithm with granularity four).

Let $N = 2^n = G^q$, $G = 2^g$, $g\,q = n$, where G is granularity.

In the general case the sequence of g shuffles of an array can be viewed as a single act with the following properties:

- The source array is divided into G equal parts.
- The objects subjected to shuffling form groups, each group containing G objects (one object from one initial group) and indices which differ in their g most significant digits (all the other digits are equal).
- The output indices (after shuffling) can be obtained by means of left g-digit cyclic rotation of the respective input indices.

In this connection it is convenient to use the term *perfect shuffle rank*, which is equal, by definition, to the equivalent number of standard perfect shuffles (*PS* of rank 1).

Omega-similar Network

In the case of a perfect shuffle of rank g, the data flow can be represented using a multistage interconnection network, similar to Omega network (*Omega-similar* network), with the following properties.

- The network has $G^q = N$ inputs and so many outputs;
- Between them there are $\log_G N = q$ number of stages, each stage containing N/G switching elements (crossbar switches of $G'G$ dimension);
- Every crossbar switch has G inputs (0 to G-1) and so many outputs;
- Each stage of the network performs a *perfect shuffle* of rank g;
- The interconnection between the consecutive stages is accomplished on the basis of coincidence of the input and output identifiers. The input identifiers include the input number as the most significant part, followed by the switch number. The output identifiers include the switch number as the most significant part, followed by the output number;
- The output identifiers of a given switch can be obtained by means of left, g-digit cyclic rotation of the respective input identifiers.

For identification of such an *Omega-similar* network we can use the term *Omega-similar* network *class*, accepting by definition the *class* of this network to be equal to the *rank* of the perfect shuffle which is performed by the network.

Classes of Indirect Hypercubes

Horizontal reduction is obtained by using one stage to perform the functions of all the stages. This is achieved by taking one stage of the network and connecting the outputs of this stage to the inputs of the same stage in accordance with the presented manner of interconnection. The resulting structure can be called single-stage indirect Hypercube of a class equal to the class of the *Omega-similar* network, from which it is obtained (in this case – class g).

The next step is parametric vertical reduction with respect to the processors P of the last structure.

Let $P = 2^p$, $P \leq N/G$.

The generalized crossbar switch (GCS) of $G'G$ dimension (G inputs and G outputs) is an integrated unified object, including a set of ordinary G´G crossbar switches whose p least significant binary digits of their numbers coincide (GCS number). GCS is considered as a complex structure which includes one simple $G'G$ crossbar switch, performing the functions of a processor element (performing a P-point FFT), and one G-port memory which serves for reading of input data for *PE* and storing of intermediate or final results.

Each crossbar switch (processor) is assigned a number which represents a p-digit binary number. With each processor indexes are associated, whose least significant digits coincide with the number of the processor.

From the global identifier (index) two fields are extracted – a group and a local identifier.

The Group identifier (GI) includes the least significant $p + g$ binary digits of the global identifier.

- The Group Identifiers are of two types – input and output.
- The input GI include the input number, as the most significant part, followed by the switch number.
- The output GI include the switch number as the most significant part, followed by the number of the output.
- The output GI of a given output is obtained by left, g-digit cyclic rotation of the respective input GI.
- The Group Identifiers identify the groups of signals that enter/exit the switch, and are used to connect the switches on the base of coincidence of the input and output GI.

The Local Identifier (LI) includes the most significant $n - p$ digits of the global identifier.

- LI identifies the signals inside the switch.
- LI most significant g digits indicate the PE input number.
- LI least significant g digits, which coincide with the most significant g digits of the input group identifier, indicate the GCS input number.
- The main property of the local identifier is a left g-digit cyclic rotation on transitions from a stage to a stage.
- In any switch (processor unit) FFT is performed on G points. Depending on the perfect shuffle presentation inside the switch, the operations can be based on radix-2 (composition of rank 1 shuffles) or a higher radix, the maximum radix being G (one perfect shuffle of rank g).
- The scheme can be regarded as a scheme of parametric realization of transpose algorithm with granularity G with a corresponding radix.

As an example we shall review the data flow analysis

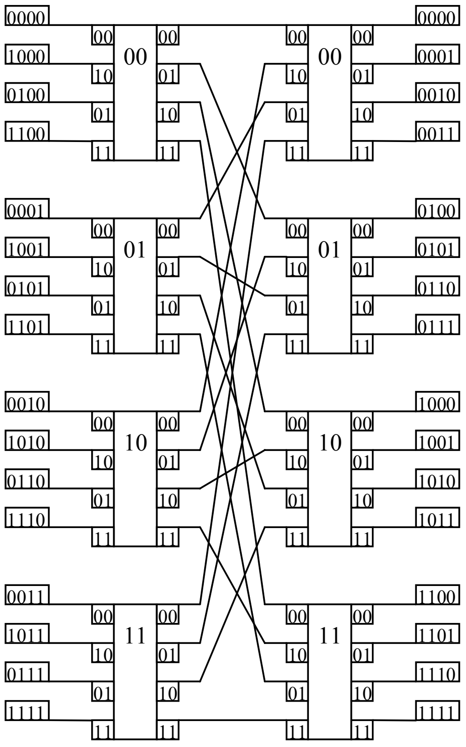

Figure 3. 16 pt *Omega-similar* class 2 network

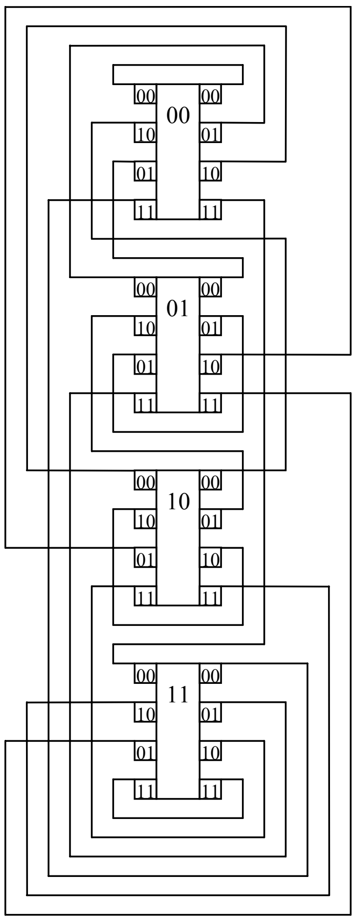

Figure 4. Indirect class 2 hypercube

of a transpose algorithm with granularity four.

Model of *Omega-like* Network and Indirect Hypercube of Class Two

For this case we have $N = 2^n = 4^{n/2}$, $P = 2^p \leq N/4$.

The interconnection network that represents the flow of data contains $\log_4 N$ stages, each stage contains $N/4$ switching elements, each with dimensions 4´4. The numbers of inputs/ outputs are represented by two-digit binary numbers. The output identifiers are obtained by 2-digit left cyclic rotation of the respective input identifiers. *Figure 3* presents the interconnection network (*Omega-similar* network of class 2) that shows the flow of data for a transpose algorithm with granularity four performing 16 pt. FFT, and *figure 4* - the respective indirect hypercube of class 2.

The vertical contraction with respect to P is achieved with the help of the previously described procedure by replacing G by 4 and g by 2 respectively.

The model considered so far can be used to implement a transpose type algorithm with granularity four. The options in terms of the basis are radix-2 and radix-4. Option radix-2 implies the presentation of shuffling in the switches (which is of rank 2) as a composition of two standard shuffles (rank one). Radix-4 variant implies a shuffle of rank two.

Radix-4 FFT

Usually radix-4 operations are called *dragonflies*, and

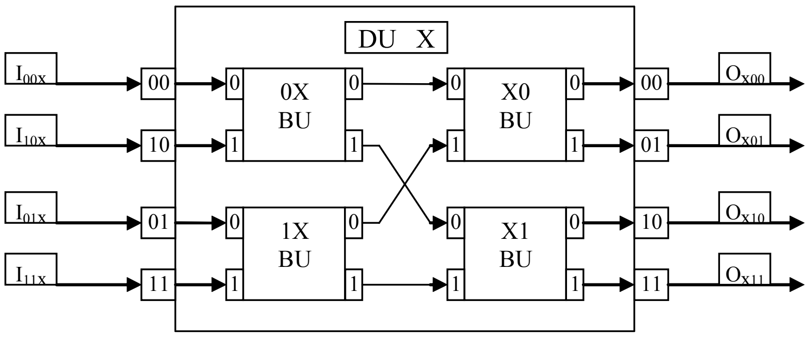

Figure 5. Dragonfly operation model

Table 1

DIT		DIF	
BU 0X	**BU X0**	**BU 0X**	**BU X0**
$P = I_{00x} + W^{2\arg} I_{10x}$ $Q = I_{00x} - W^{2\arg}.I_{10x}$	$O_{x00} = P + W^{\arg} R$ $O_{x01} = P - W^{\arg}.R$	$P = I_{00x} + I_{10x}$ $Q = (I_{00x} - I_{10x})\, W^{2\arg}$	$O_{x00} = P + R$ $O_{x01} = (P - R)\, W^{\arg}$
BU 1X	**BU X1**	**BU 1X**	**BU X1**
$R = I_{01x} + W^{2\arg} I_{11x}$ $S = I_{01x} - W^{2\arg} I_{11x}$	$O_{x10} = Q + W^{\arg} e^{j\pi/2} S$ $O_{x11} = Q - W^{\arg}.e^{j\pi/2} S$	$R = I_{01x} + I_{11x}$ $S = (I_{01x} - I_{11x})\, W^{2\arg}$	$O_{x10} = Q + S$ $O_{x11} = (Q - S).W^{\arg}.e^{j\pi/2}$

they implement the operations of four radix-2 butterflies. *Figure 5* shows a model of dragonfly operation based on butterfly units (BU). Specified BU numbers are the numbers that they would have in a Omega network.

Table 1 indicates the radix-2 DIT/DIF butterfly operations.

Realized in this way (as a combination of four butterfly operations), one dragonfly operation requires four complex multiplications and eight complex additions.

After processing the expressions from *table 1*, DIT/DIF versions of radix-4 dragonfly operations are obtained as follows:

DIT

$$\begin{aligned} &O_{x00} = I_{00x} + W^{2\arg} I_{10x} + W^{\arg} I_{01x} + W^{3\arg} I_{11x} \\ (4.a)\quad &O_{x01} = I_{00x} + W^{2\arg} I_{10x} - (W^{\arg} I_{01x} + W^{3\arg} I_{11x}) \\ &O_{x10} = I_{00x} - W^{2\arg} I_{10x} + e^{j\pi/2}(W^{\arg} I_{01x} - W^{3\arg} I_{11x}) \\ &O_{x11} = I_{00x} - W^{2\arg} I_{10x} - e^{j\pi/2}(W^{\arg} I_{01x} - W^{3\arg} I_{11x}) \end{aligned}$$

DIF

$$\begin{aligned} &O_{x00} = I_{00x} + I_{10x} + I_{01x} + I_{11x} \\ (4.b)\quad &O_{x01} = (I_{00x} + I_{10x} - I_{01x} - I_{11x})\, W^{\arg} \\ &O_{x10} = (I_{00x} - I_{10x} + I_{01x} - I_{11x}) W^{2\arg} \\ &O_{x11} = (I_{00x} - I_{10x} - I_{01x} + I_{11x})\, e^{j\pi/2}\, W^{3\arg} \end{aligned}$$

Systems (4.a),(4.b) require three complex multiplications and eight complex additions, thus saving one complex multiplication. This is the reason to use higher radices.

Generalization of the Concept for an Indirect Hypercube with Respect to Real-valued FFT/FHT

Real-valued FFT

When the initial sequence $X(n)$ in (1.a) is real-valued, then:

a) For $k = 0$, $F(0)$ and $F(N/2)$ are real numbers and

$$F(0)real = \sum_{n=0}^{N-1} X(n) \qquad F(0)imag = 0$$

(5.a)

$$F(N/2)real = \sum_{n=0}^{N-1} (-1)^n X(n) \qquad F(N/2)imag = 0$$

b) For $0 < k < N/2$, $F_{(N-k)} = F_{(k)}{}^*$ (conjugated complex numbers), and

$$F(\mathrm{k})real = \sum_{n=0}^{N-1} X(\mathrm{n}).Cos(\frac{2\pi}{N})kn \qquad F(\mathrm{k})imag = \sum_{n=0}^{N-1} X(\mathrm{n}).Sin(\frac{2\pi}{N})kn$$

(5.b)

$$F(N-\mathrm{k})real = \sum_{n=0}^{N-1} X(\mathrm{n}).Cos(\frac{2\pi}{N})kn \quad F(N-\mathrm{k})imag = -\sum_{n=0}^{N-1} X(\mathrm{n}).Sin(\frac{2\pi}{N})kn$$

When the initial sequence $X(n)$ is complex conjugated, then:

a) The inputs $X(0)$ and $X(N/2)$ are real numbers.

b) For $0 < k < N/2$, the inputs $X_{(N-k)} = X_{(k)}{}^*$ (conjugated complex numbers).

c) The outputs are real numbers.

From (5.a) and (5.b) it is very well seen how the information in this case is doubled.

As above mentioned, efficient FFT algorithms have been designed for this situation. One approach consists in taking an ordinary FFT algorithm (e.g. Cooley-Tukey) and removing the redundant parts of the computation, saving roughly a factor of two in time and memory [Bergland-1]. Alternatively, it is possible to express an *even*-length real-input DFT as a complex DFT of half the length (whose real and imaginary parts are the even/odd elements of the original real data), followed by $O(N)$ post-processing operations. The two mentioned approaches can be combined.

Generally, for the case of *real-valued* FFT (direct and inverse) the radix-2 DIT FFT as more suitable is used as a base, and for the case of conjugation symmetrical FFT (direct and inverse), the radix-2 DIF FFT is used as a base.

Combining symmetrical butterflies, removing redundances and making some substitutions, three types of butterflies are obtained for each of the systems (2.a) and (2.b) (*table 2*).

FHT

DHT is defined according to the following formula:

$$(6)\quad H_k = \sum_{n=0}^{N-1} X_n[cos(\frac{2\pi}{N}nk) + sin(\frac{2\pi}{N}nk)],\ k = 0,\ldots, N\text{-}1$$

Usually cos(z) + sin(z) is denoted by cas(z).

Table 2

DIT Stages 0 to *n*	DIF Stages 0 to *n*
Type A (stage 0, k = 0) $X_{f(0)} = X_{f0(0)} + X_{f1(0)}$ $X_{f(N/2)} = X_{f0(0)} - X_{f1(0)}$ $X_{f(N/4)} = X_{f0(N/4)} + X_{f1(N/4)}$ $X_{f(3N/4)} = X_{f0(N/4)} - X_{f1(N/4)}$	Type C (stage n, k = 0) $X_{f(0)} = X_{f0(0)} + X_{f0(N/4)}$ $X_{f(N/2)} = X_{f0(0)} - X_{f0(N/4)}$ $X_{f(N/4)} = X_{f1(0)} + X_{f1(N/4)}$ $X_{f(3N/4)} = X_{f1(0)} - X_{f1(N/4)}$
Type B.0 (stage >0, k = 0) $X_{f(0)} = X_{f0(0)} + X_{f1(0)}$ $X_{f(N/2)} = X_{f0(0)} - X_{f1(0)}$ $X_{f(N/4)} = X_{f1(N/4)}$ $X_{f(3N/4)} = X_{f0(N/4)}$	Type D.0 (stage < n, k = 0) $X_{f(0)} = X_{f0(0)} + X_{f0(N/4)}$ $X_{f(N/2)} = X_{f0(0)} - X_{f0(N/4)}$ $X_{f(N/4)} = 2.X_{f1(N/4)}$ $X_{f(3N/4)} = 2.X_{f1(0)}$
Type B.1 (stage >0, k > 0) $X_{f(k)} = X_{f0(k)} + \cos((2\pi/N)k).X_{f1(k)} + \sin((2\pi/2N)k).X_{f1(N/2-k)}$ $X_{f(N-k)} = X_{f0(N/2-k)} + \cos((2\pi/N)k).X_{f1(N/2-k)} - \sin((2\pi/N)k).X_{f1(k)}$ $X_{f(N/2+k)} = X_{f0(k)} - \cos((2\pi/N)k).X_{f1(k)} - \sin((2\pi/N)k).X_{f1(N/2-k)}$ $X_{f(N/2-k)} = X_{f0(N/2-k)} - \cos((2\pi/N)k).X_{f1(N/2-k)} + \sin((2\pi/N)k).X_{f1(k)}$	Type D.1 (stage < n, k > 0) $X_{f(k)} = X_{f0(k)} + X_{f1(k)}$ $X_{f(N-k)} = X_{f0(N/2-k)} + X_{f1(N/2-k)}$ $X_{f(N/2+k)} = (X_{f0(k)} - X_{f1(k)}).\cos((2\pi/N)k) + (X_{f0(N/2-k)} - X_{f1(N/2-k)}).\sin((2\pi/N)k)$ $X_{f(N/2-k)} = (X_{f0(N/2-k)} - X_{f1(N/2-k)}).\cos((2\pi/N)k) - (X_{f0(k)} - X_{f1(k)}).\sin((2\pi/N)k)$

Generally, FHT is based on the already mentioned *Danielson-Lanczos lemma* [Danielson-1] – the base of the Cooley-Tukey radix-2 FFT algorithm.

FHT splits the *N*-point sequence $X(n)$ into two smaller ($N/2$)-point sequences $X_0(n)$ (even) and $X_1(n)$ (odd). The two swquences are Hartley transformed into $Xh_0(k)$ and $Xh_1(k)$ and combined into $X_h(k)$ in accordance with the next relations:

$$X_h(k)=\begin{cases} Xh_{0(k)}+\cos((2\pi/N)k)Xh_{1(k)}+\sin((2\pi/N)k)Xh_{1(N/2-k)}, & \text{for } 0\le k\le N/2-1 \\ Xh_{0(k-N/2))}-\cos((2\pi/N)(k-N/2)\ Xh_{1(k-N/2))} - \sin((2\pi/N)(k-N/2)\ Xh_{1(N-k)} & \text{for } N/2 \le k \le N-1. \end{cases} \quad (7)$$

Equations (7) result in the so called *double FHT butterfly* (DFHTB)[Ulman-1] which are of two types and are similar to the RFFT butterflies shown in Table 1. For example, a DIT version of DFHTB can be obtained by replacement of type B.0 butterfly with type A butterfly in DIT column of *table 2*. Alternatively, a DIF version of DFHTB can be obtained by replacement of type D.0 butterfly with type C butterfly and changing the direction of rotation in type D.1 butterfly in the DIF column of *table 2*.

RFFT/FHT Parallel Implementation

Analyzing the butterflies from *table 2* it is not difficult to see that:

• The inputs (respectively outputs) form pairs which come from (respectively go to) one and the same butterfly. For example, the input pairs are $(X_{f0(0)}, X_{f0(N/4)})$, $(X_{f0(k)}, X_{f0(N/2-k)})$, etc,, the output pairs are $(X_{f(0)}, X_{f(N/2)})$, $(X_{f(k)}, X_{f(N-k)})$, etc.

• Butterflies of type B.1 (respectively D.1) are equivalent to the respective FFT butterflies and butterflies of other types are near to the respective FFT butterflies.

This is the reason why one of the most popular approaches in the creation of RFFT/FHT algorithms is the imitation of the respective FFT algorithm. This approach includes:

• Presenting the *N*-point real-valued RFFT/FHT array as an *N*/2-point complex array (each position in the array contains two components – *real* and *imaginary*). Such a *complex* pair contains either two real values (e.g. $(X_{f(0)}, X_{f(N/2)})$), or the real and imaginary parts of two complex-conjugated values of the original FFT algorithm.

• Replacing the original FFT butterflies by the butterflies presented in *table 2* and control adjustment.

Although this approach seems very attractive, when applied to the implementation of parallel algorithms, it raises certain problems. Parallel algorithms require regularity in the structure of their signal graph (flow of data).

Each RFFT butterfy incorporates two FFT butterflies with conjugated transform parameters (e.g. (k, $N/2$-k), (0, $N/4$), etc.). For each RFFT butterfly we have to choose one of two conjugated (complementary) values and on this base to identify the respective complex pair. Choosing $0\le k<N/4$ does not ensure the necessary structural regularity.

For *N*-point FFT ($N=2^n$) identifiers there are *n*-bit binary numbers. Presenting the *N*-point real-valued array as a *N*/2-point complex array makes the identifiers *n*-1 binary numbers.

So, creating a parallel RFFT/FHT algorithm requires the solution of the next two problems:

- Identification of the complex pairs.
- Reduction of the n-bit identifiers into n-1 bit identifiers.

Parallel implementations of FHT/RFFT have been made for systolic arrays [Marchesi-1] [Chang-1], hypercuboid multicomputers [Zapata-1], as well as VLSI implementations [Zapata-2], [Liu-1]. The proposed solutions to the problems indicated are suitable mainly for the realization of algorithms of *data-exchange* type.

FFT Identifier Analysis

It was already shown that the structural regularity (inherent parallelism) in Cooley-Tukey FFT algorithm is due to the *perfect shuffle* method, and the identifiers (indices) have certain properties that support this method.

It was also shown that radix-2 DIT/DIF FFT, is based on the already mentioned *Danielson-Lanczos lemma* – formaton of two N-point sequences ($X_{f(k)}$ and $X_{f(N/2+k)}$) on the base of two (even and odd) $N/2$-point sequences ($X_{f0(k)}$ and $X_{f1(k)}$).

Let $N = 2^n$.

The signal identifier is a n-bit binary number and contains two fields – S field and K field. S field is the sequence number, while K field identifies the parameter k (the transform parameter).

The sequence numbers are formed in the following manner (DIT). From a N-point sequence two (N/2)-point sequences are formed – an even (0) and an odd one (1). From the even (N/2)-point sequence two (N/4)-point sequences are formed – an even (00) and an odd one (10). From the odd (N/2)-point sequence also two (N/4)-point sequences are formed – an even (01) and an odd one (11). This procedure continues up to the enumeration of all the points. This type of sequence enumeration provides two important consequences:

1. In FFT butterflies enter some signals, whose S fields are equal with the exception of their most significant digit.
2. The output signals of a given FFT butterfly have for a sequence number the common, least significant part of the sequence number of the input sequences.

The signal identifiers are organized according to the following rules:

1. S field is the most significant part of the signal identifier.
2. K field is the least significant part of the signal identifier, and it is the bit-reversed value of k parameter.

With every stage K field increases with one digit and S field decreases with one digit. So, the initial identifiers contain only S field (the point index), while the final identifiers contain only K field (the final results in a bit-reversed form).

The following two properties of the signal identifiers are very important (perfect shuffle identification):

1. In a given butterfly two signals enter, whose identifiers are equal with the exception of their most significant digit.
2. The identifiers of the output signals of a given butterfly can be obtained by means of left cyclic rotation of the identifiers of the respective input signals.

RFFT Butterfly Identifier Analysis

Let $N = 2^n$. Presenting the real-valued array as a complex array we obtain an $N/2$-point complex array. Every signal is a *RI* pair (contains two parts – *R* (*Real*) and *I* (*Imaginary*)). The *R* part contains the value of the real component of the respective conjugated FFT complex pairs and the *I* part – the respective imaginary component. The signal identifier must consist of n-1 binary digits. We can assume the same structure of the signal identifier consisting of two parts - S field and K field. S field is again the most significant part of the identifier, and K field is the least significant part. K field is the bit-reversed code of the transform parameter k. S field is the same as in FFT case. Problems arise with K field identification (we already discussed these problems). It is not clear beforehand which value of the two possible values to choose, and how to code this value in a field which has been subjected to an 1-bit reduction. As already mentioned, choosing $0 \le k < N/4$ does not ensure the necessary structural regularity.

Analysis of k-values in *RI* pairs shows that k-values in a given *RI* pair always belong to two different classes. The first class, D (direct) class, is generated by '0'-value of the parameter k. The second class, C (complementary) class, is generated by '1'-value of the parameter k. In a given *RI* pair, when the *R* part is a D-value, the *I* part is a C-value and vice versa – when the *R* part is a C-value, the *I* part is a D-value. D-values generate always D-values and C-values generate always C-values. This allows the identification to be made on the base of D-values.

The characterization of D and C classes is as follows:

$$k \in D : \{k = 0 \text{ or } k = (4p+3)2^q, \text{ for some } p, q \ge 0\},$$

(8)

$$k \in C : \{k \ne 0 \text{ and } k = (4p+1)2^q, \text{ for some } p, q \ge 0\}.$$

One-bit reduction of K-field is performed by removing the least significant '1' in the binary presentation of the given D-value when it is not zero, or by remoing one binary zero when it is zero. Arithmetically it is expressed as follows:

$$k\text{-}value = (4p+3)2^q, \text{ for some } p, q \ge 0,$$

(9)

$$k\text{-}code = (2p+1)2^q, \text{ for some } p, q \ge 0.$$

The fact that this type of identification exhibits the two mentioned properties which idenyify *perfect shuffle* is very important:

1. In a given butterfly two signals enter, whose identifiers are equal with the exception of their most significant digit.
2. The identifiers of the output signals of a given butterfly can be obtained by means of left cyclic rotation of the identifiers of the respective input signals.

This type of identification leads to a variant of the described Omega network, let us say *DC Omega network*. The difference is that *DC* Omega network (*figure 6*) possesses one stage more than the standard Omega network.

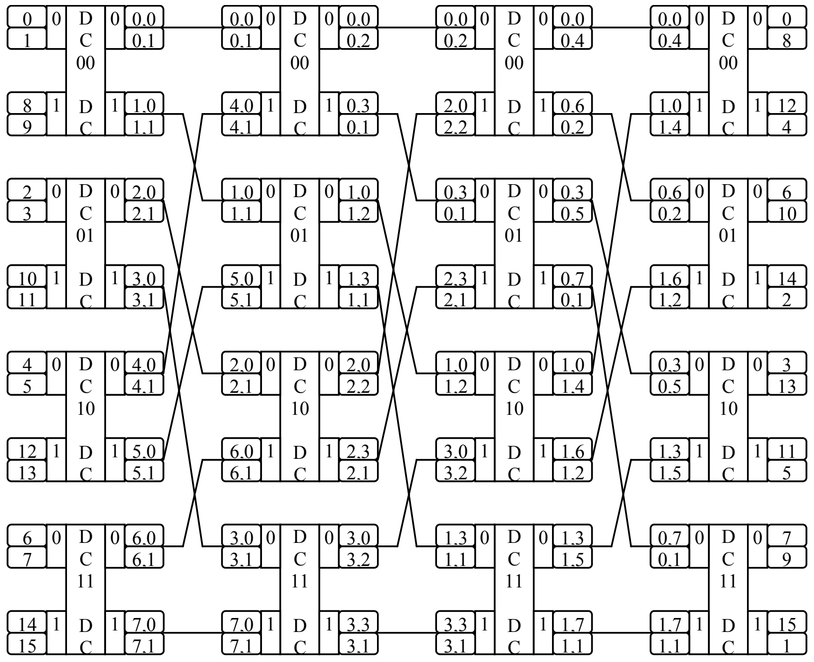

Figure 6. 16 pt. DC Omega network

This is due to the presence of one preliminary (post) stage in the FHT/RFFT algorithms. In any case, both types of networks (*Omega-like* networks and *DC Omega-like* networks) result in one and the same type of indirect hypercubes, including the parametrical synthesis of indirect hypercubes of different clases.T

he DC Omega network shown in *figure 6* presents the flow of data of a 16-point parallel (4 processors) radix-2 RFFT/FHT DIT (for RFFT, the inputs are real values) transpose algorithm with granularity two (the finest granularity). Considering the flow of data from right to left, the DIF is regarded (for RFFT, the inputs are conjugated complex numbers).

The basic features of this algorithm are as follows:

• The signals are identified by their D-value part which is the *R* component of the *imitated* complex signal.

• *R* components contain the real (even) components and *I* components contain the imaginary (odd) components of the conjugated complex FFT values.

• The initial *RI* pairs (DIT) are formed on even/odd basis – e.g. (0, 1), (2, 3), etc.

• When the initial *RI* pairs (DIT) are in a natural order, the final pairs are in a bit-reversed order of the coded D-values.

• When the initial *RI* pairs (DIT) are in a bit-reversed order, the final pairs are in a natural order of the coded D-values.

DCT

Discrete Cosine Transform (DCT) and Discrete Sine Transform (DST) are Fourier-related transforms similar to the discrete Fourier transform (DFT), but using only real numbers. DCTs and DSTs are equivalent to DFTs of roughly twice the length, operating on real data, with even symmetry (DCT) or odd symmetry (DST), where in some variants the input and/or output data are shifted by half a sample.

DCTs and DSTs are widely employed in solving partial differential equations (PDE) by spectral methods, where the different variants of the DCT/DST correspond to slightly different even/odd boundary conditions at the two ends of the array.

The most common variant of the discrete cosine transform is the type-II DCT [Ahmed-1], which is often called simply the DCT; its inverse, the type-III DCT, is correspondingly often called simply the inverse DCT or the IDCT.

The DCT, and in particular the DCT-II, is often used in signal and image processing, especially for lossy data compression, because it has a strong “energy compaction” property [Ahmed-1], [Rao-1]: most of the signal information tends to be concentrated in a few low-frequency components of the DCT, approaching the Karhunen-Loève transform (which is optimal in the decorrelation sense) for signals based on certain limits of Markov processes.

Although the direct computation of DCT would require O(N^2) operations, it is possible to compute it with O(N log N) complexity similarly to the fast Fourier transform (FFT). These algorithms, with O(N log N) complexity, are known as fast cosine transform (FCT) algorithms.

DCTs can be computed also via FFTs combined with O(N) pre- and post-processing steps.

Specialized FCT algorithms have been designed for this purpose – e.g. by taking an FFT (e.g. Cooley-Tukey FFT) and eliminating the redundant operations.

There are applications that require parallel implementation of DCT.

Specialized FCT algorithms, however, have an irregular computational structure, which makes their efficient parallelization a very difficult task. Thus, in practice, it is often easier to obtain high performance for DCT with FFT-based algorithms.

DFT, RDFT and DCT

DCT-II

Consider the following transform:

(10.a) $C_k = F(k)real.\cos[\frac{\pi k}{2N}] + F(k)imag.\sin[\frac{\pi k}{2N}]$, where

$$F(k) = F(k)real + iF(k)imag = \sum_{m=0}^{N-1} X(n)\,\mathrm{W}^{-kn}$$

as a DFT of the real valued input sequence $X(n)$ in accordance with (1.a), (5.a) and (5.b).

After some transformations (10.a) is converted as follows:

$$C_k = \sum_{n=0}^{N-1} X(n)\cos[\frac{2\pi kn}{N}].\cos[\frac{\pi k}{2N}] - \sum_{n=0}^{N-1} X(n)\sin[\frac{2\pi kn}{N}].\sin[\frac{\pi k}{2N}] = \sum_{n=0}^{N-1} X(n)\cos[\frac{\pi k(4n+1)}{2N}]$$

After some transformations (10.a) is converted as follows:

(10.b) $C_k = \sum_{n=0}^{N-1} Y_m \cos[\frac{\pi}{2N}(2m+1)k]$, where

$m = 2n$ for $n = 0,\dots, N/2-1$;

(10.c) $m = 2(N-n)-1$ for $n = N/2,\dots, N-1$.

$Y_m = X_n$.

(10.b) is the definition for DCT II.

DCT-III

Let $X_0,\dots, X_{N-1}$ be a sequence of real numbers , and let Y be a complex-conjugated input sequence of the following form:

$Y_0 = X_0$ is a real number.

$$Y_{m\_real} = X_m \cos[\frac{\pi m}{2N}] + X_{N-m}\sin[\frac{\pi m}{2N}]$$

(11.a) $$Y_{m\_imag} = X_m \sin[\frac{\pi m}{2N}] - X_{N-m}\cos[\frac{\pi m}{2N}]$$

$$Y_{N-m\_real} = X_{N-m}\sin[\frac{\pi m}{2N}] + X_m\cos[\frac{\pi m)}{2N}] = Y_{m\_real}$$

$$Y_{N-m\_imag} = X_{N-m}\cos[\frac{\pi m}{2N}] - X_m\sin[\frac{\pi m}{2N}] = -Y_{m\_imag}$$

$$Y_{N/2\_real} = 2X_{N/2}\cos[\frac{\pi}{4}] = Y_{N/2}$$

For the case of inverse DFT we obtain

(11.b) $F(k) = X_0 + 2\sum_{m}^{N-1} X_m cos[\frac{\pi m(4k+1)}{2N}]$

After some transformations (11.b) is converted as follows:

(11.c) $$C(q) = \frac{1}{2}X_0 + \sum_{m=1}^{N-1} X_m \cos[\frac{\pi m(2q+1)}{2N}],$$

where

(11.d) $q = 2k$ for $k = 0,\dots, N/2-1;$
$q = 2(N-n)-1$ for $k = N/2,\dots, N-1.$
$C_q = F_k.$

DCT III is defined by (11.c).

FCT

From the provided analysis we see the important role which real-valued DFT plays for DCT. Once the problem with real-valued DFT has been solved, it is clear that the indirect hypercube concept can be applied for DCT.

Adequate FCT for DCT II can be achieved by:

- Reordering of the input data in accordance with (10.c).
- Adding one additional module which can be combined with the last RFFT stage – case of real-valued inputs (DIT case) (*table 2* and *table 3* – 1st column).

Adequate FCT for DCT III can be achieved by:

- Adding one additional module which can be combined with the first RFFT stage – case complex-conjugated inputs (DIF case) (*table 2* and *table 3* – 2nd column).
- Reordering of the output data (results) in accordance with (11.d).

The described approach permits implementation of the *indirect hypercube concept* for DCT. So, the presented on *figure 6* "16 pt. DC Omega network" shows also (except 16 point RFFT/FHT) the flow of data of 16 point FCT II (from left to right) and 16 point FCT III (from right to left).

On the basis of indirect hypercubes with low granularity (two and four) fast FCT modules can be designed for real time applications, and on the basis of indirect hypercubes with high granularity parallel FCT transpose algorithms can be designed for the needs of computational mathematics.

Some Applications

The discussed models of indirect hypercubes can be used for different type of concrete implementations of parallel algorithms of transpose type with different granularity/radix.

Fast FFT Processors

Conventional FFT hardware architectures include trade-offs among complexity, power consumption, die size, and other similar parameters. However, these architectures do not have the scalability to meet the high speed demands of the FFT processor for the new high data rate wireless technologies in communication (OFDM-based technologies like ADSL, VDSL, etc.), high-speed FFT-based 2D image processing and others, demanding high-speed FFT processors.

The advance in technologies enables the development of architectures, optimal for a given class of algorithms, including FFT. Recently considerable efforts have been dedicated for FPGA-based parallel realizations of al-

Table 3. Additional FCT butterflies

DIT (real-valued inputs) ***FCT II combined last stage***	***DIF (complex-conjugated inputs)*** ***FCT III combined first stage***
Type E.0 (k = 0) $C_{(0)} = X_{f0(0)} + X_{f1(0)}$ $C_{(N/2)} = (X_{f0(0)} - X_{f1(0)}).\cos(\pi/4)$ $C_{(3N/4)} = X_{f0(N/4)}.\cos(3\pi/8) + X_{f1(N/4)}.\sin(3\pi/8)$ $C_{(N/4)} = X_{f0(N/4)}.\sin(3\pi/8) - X_{f1(N/4)}.\cos(3\pi/8)$	*Type F.0 (k = 0)* $X_{f0(0)} = X_{(0)} + {}_2X_{(N/2)}.\cos(\pi/4)$ $X_{f1(0)} = X_{(0)} - 2X_{(N/2)}.\cos(\pi/4)$ $X_{f0(N/4)} = 2(X_{(3N/4)}.\cos(3\pi/8) + X_{(N/4)}.\sin(3\pi/8))$ $X_{f1(N/4)} = 2(X_{(3N/4)}.\sin(3\pi/8) - X_{(N/4)}.\cos(3\pi/8))$
Type E.1 (k > 0) $C_{(k)} = (X_{f0(k)} + \cos(2\pi k/N).X_{f1(k)} + \sin(2\pi k/N).X_{f1(N/2-k)}).\cos(\pi k/2N) + (X_{f0(N/2-k)} + \cos(2\pi k/N).X_{f1(N/2-k)} - \sin(2\pi k/N).X_{f1(k)}).\sin(\pi k/2N)$ $C_{(N-k)} = (X_{f0(k)} + \cos(2\pi k/N).X_{f1(k)} + \sin(2\pi k/2N).X_{f1(N/2-k)}).\sin(\pi k/2N) - (X_{f0(N/2-k)} + \cos(2\pi k/N).X_{f1(N/2-k)} - \sin(2\pi k/N).X_{f1(k)}).\cos(\pi k/2N)$ $C_{(N/2+k)} = (X_{f0(k)} - \cos(2\pi k/N).X_{f1(k)} - \sin(2\pi k/N).X_{f1(N/2-k)}).\cos(\pi/4 + \pi k/2N) + (X_{f0(N/2-k)} - \cos(2\pi k/N).X_{f1(N/2-k)} + \sin(2\pi k/N).X_{f1(k)}).\sin(\pi/4+\pi k/2N)$ $C_{(N/2-k)} = (X_{f0(k)} - \cos(2\pi k/N).X_{f1(k)} - \sin(2\pi k/N).X_{f1(N/2-k)}).\sin(\pi/4 + \pi k/2N) - (X_{f0(N/2-k)} - \cos(2\pi k/N).X_{f1(N/2-k)} + \sin(2\pi k/N).X_{f1(k)}).\cos(\pi/4 + \pi k/2N)$	*Type F.1 (k > 0)* $X_{f0(k)} = X_{f(k)} + X_{f(N/2+k)}$ $X_{f0(N/2-k)} = X_{f(N-k)} + X_{f(N/2-k)}$ $X_{f1(k)} = (X_{f(k)} - X_{f(N/2+k)}).\cos(2\pi k/N) - (X_{f(N-k)} - X_{f(N/2-k)}).\sin(2\pi/N)$ $X_{f1(N/2-k)} = (X_{f(N-k)} - X_{f(N/2-k)}).\cos(2\pi k/N) + (X_{f(k)} - X_{f(N/2+k)}).\sin(2\pi/N)$ $X_{f(k)} = X_{(k)}.\cos(\pi k/2N) + X_{(N-k)}.\sin(\pi k/2N)$ $X_{f(N-k)} = X_{(k)}.\sin(\pi k/2N) - X_{(N-k)}.\cos(\pi k/2N)$ $X_{f(N/2+k)} = X_{(N/2+k)}.\cos(\pi/4 + \pi k/2N) + X_{(N/2-k)}.\sin(\pi/4 + \pi k/2N)$ $X_{f(N/2-k)} = X_{(N/2+k)}.\sin(\pi/4 + \pi k/2N) - X_{(N/2-k)}.\cos(\pi/4 + \pi k/2N)$

gorithms of this class. FPGA provide highly improved performance and capacity, a number of integrated specialized functions (embedded multipliers, distributed and block RAMs, specialized DSP slices, etc.), as well as flexibility, shorter design cycles and lower development costs. The design process is shortened and facilitated also by the usage of high-level languages for hardware description, such as Verilog, VHDL, etc.

VHDL possesses some advantages, important for the design, simulation and testing of complex systems. It is a universal tool for description, covering almost all development stages - programming, simulation, verification, synthesis and documentation. It permits the design and simulation of single-layer and multilayer structures, providing the possibility for an arbitrary level of detailing, as well as the parameterization of any type of characteristics.

Modern fast FFT processors, based on FPGA technology, have the following features:

- Multiple processing elements performing the function of butterfly units (radix-2) or dragonfly units (radix-4), internal linear pipeline structure, which allows them to give a result on every cycle of the system clock, and working in series (cascaded) or in parallel.
- Data format - fixed point 2's complement.
- Scaling strategy to avoid overflow.
- Synchronous SIMD array architecture.

The analysis shows that the low class (e.g. class 1 and 2) indirect hypercube topology is very suitable for realization of fast FFT/FHT/RFFT/FCT processors for real-time applications. It enables optimization of the resource utilization, ensures low latency and high efficiency – the main requirements for such systems.

As an example we shall describe two SIMD array architectures based on the indirect hypercube concept and implementing FFT, respectively RFFT/FHT/FCT algorithms of a transpose type with fine granularity (two and four).

Parallel SIMD Array Architecture for Radix-2 FFT/RFFT/FHT/FCT

This architecture implements the concept of class 1 indirect hypercube. It consists of Generalized Crossbar Switches performing the function of processor blocks (PB). PB are connected between each other in a *perfect shuffle* interconnection manner based on matching the input and output group identifiers. So, every PB is connected to two PBs for sending data and to two PBs for receiving data.

The processor block (*figure 7*) includes a simple 2´2 crossbar switch, acting as a processing element (PE), performing the functions of a butterfly unit, and two dual-port memories (DM) for storing initial data, intermediate and final results (the doubling of DM number is necessary because of the pipeline structure of the PE unit). For a given stage of the FFT transform one of these two DMs is used by PE as a source of data for the current butterfly operations and the other DM is used for storing the results of the current butterfly operations of the other PBs (two),

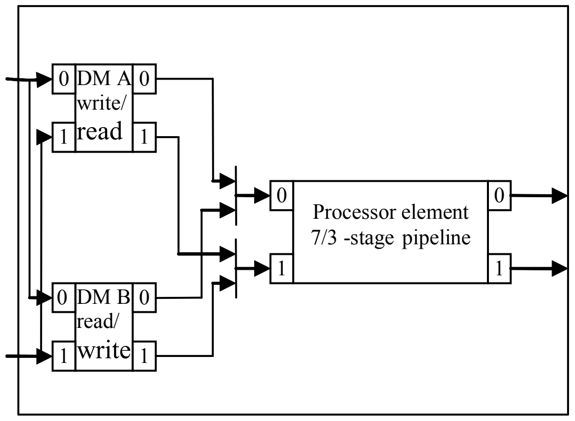

Figure 7. Radix-2 processor block

which are connected with the given PB. DMs alternatively change their role on a stage basis.

PE is realized as a 7-stage linear pipeline. It computes radix-2 butterflies and yields a result every cycle.

The input data, as well as the intermediate results are presented in a fixed point, 2's complement format. Twiddle factors (unique for different PEs) are held in coefficient look-up tables (CLUT). From coefficients viewpoint, the stages are divided into two classes: low stages and high stages. The low stages are the first n_1 stages. The high stages are the remaining n_2 stages. For the low stages every PE needs one coefficient pair (e.g. sin*x* and cos*x*) per a stage. For the high stages every PE needs totally 2^{n2-1} coefficient pairs. The set of coefficients for the high stages of a given PE includes all the coefficients, whose arguments have, as most significant digits of their binary presentation, the bit reversed number of PB. For RFFT/FHT/FCT only D-values are generated. Two variants of complex multiplication are possible – standard (4 multiplications/2 additions) and nonstandard (3 multiplications/5 additions). A block-floating point is implemented for dealing with arithmetic overflows.

DM addressing is in accordance with the *Omega approach* and is based on the main property of the local identifier – left/right cyclic rotation on transitions from a stage to a stage in case of a natural/bit-reversed order of the initial data. Four address sequences are generated – two read and two write address sequences. The write address sequences are obtained through left/right cyclic rotation of the respective read address sequences. The addresses of the low parts of CLUTs follow the stage number. The algorithm for generation of the address sequence for the high parts of CLUTs is similar to the respective algorithm in the scalar case (one-processor system). The addresses of RFFT/FHT CLUTs are generated on the base of the codes (9) of the respective D-values.

Parallel SIMD Array Architecture for Radix-4 FFT/RFFT/FHT

This architecture implements the concept of class 2 indirect hypercube. It consists also of Generalized Crossbar Switches performing the function of processor blocks. PBs are connected between each other in a *perfect shuffle rank-2* interconnection manner based on matching the input and output group identifiers. So, every PB is connected to min(P, 4) PBs for sending data and to min(P, 4) PBs for receiving data, where P, P=2^p, is the total number of PBs.

The main differences in the organization of processor blocks for radix-4 systems, compared with the organization of PB for radix-2 systems are in the following items:

- Organization of the data memory.
- Organization of PE.

It is known that one of the basic problems connected with the realization of fast FFT processors are the memory conflicts. In the case of radix-2 systems these problems are solved with the help of dual-port memories. In the case of radix-4 systems it is necessary to use quad-port memories for solving these problems. The use of such memories in a pure form poses additional problems and disadvantages of technological nature. The problem here is solved by an alternative approach – configuring one quad-port memory using four dual-port memories.

A starting point for the configuration of QM is the local identifier (LI). The two most significant digits of LI point to the QM reading port and the two least significant digits of LI show the QM writing port. During the read/write phases from QM, read/written four data items are read, whose local identifiers differ only in the numbers of the read/write ports. This means that the four data items that have been stored in QM during the write phase through various ports afterwards are read from one and the same port of QM during the next reading phase. This is the essence of the problem "memory conflicts" which requires the use of multi-port memories – quad-port memories in radix-4 case.

These specific features of QM usage offer a possibility for configuring a QM by means of four DMs on the base of the following rules (*figure 8*):

- Each DM receives a two-digit binary number (identifier).
- One of the digits of these identifiers (e.g. the most significant) together with the DM write port number specifies the QM write port number.
- The other digit (e.g. the least significant) together with the DM read port number specifies the QM read port number.
- DM selection for read/write operations, as well as data multiplexing during a read phase is accomplished with the help of the respective digits of the local identifier.

This approach can be applied for configuring a dual-port memory on the base of four standard (one-port) memories.

The processor block (*figure 9*) includes a simple 4´4 crossbar switch acting as a processing element (PE), performing the functions of a dragonfly unit, and two quad-port memories (QM) for storing the initial data, intermediate and final results. For a given stage of the FFT transform, one of these two QMs is used by PE as a source of data for the current dragonfly operations and the other QM is used for storing the results of the current dragonfly operations of the other PBs, which are connected with the given PB. QMs alternatively change their role on a stage basis.

PE is realized as a 8-stage linear pipeline.

Twiddle factors (unique for different PEs) are also held in coefficient look-up tables. For each PE three sets of tables are generated - for arguments arg, 2.arg and 3.arg (4.a/b). As in the case of radix-2, each table consists of two parts, called lower and upper. The lower part contains the coefficients needed for the lower stages, and the upper – for the upper stages. For RFFT/FHT/FCT only D-values are generated. The block-floating point is implemented for dealing with arithmetic overflows.

QM addressing is in accordance with the *Omega approach* and is based on the main property of the local

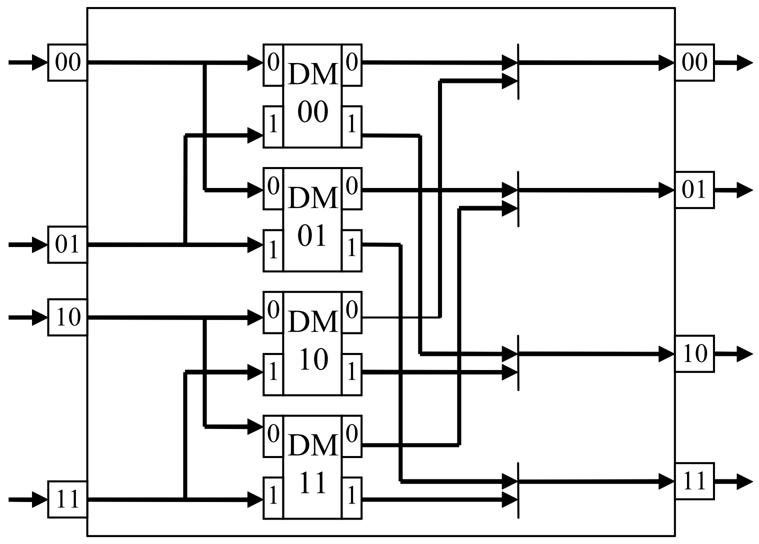

Figure 8. Configuring of QM using four DMs

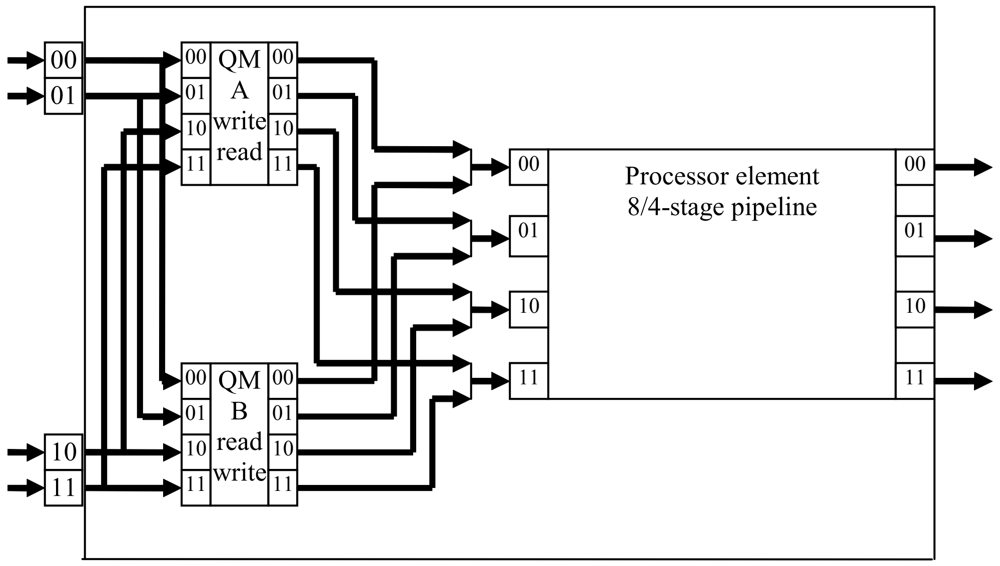

Figure 9. Radix-4 processor block

identifier – left/right 2-digit cyclic rotation on transitions from a stage to a stage in the case of a natural/bit-reversed order of the initial data. Eight address sequences are generated – four read and four write address sequences. The write address sequences are obtained through left/right 2-digit cyclic rotation of the respective read address sequences.

Implementations

Three types of implementation are possible. The 1-phase architecture performs the 3 phases (data input, data transform and data output) sequentially. The 2-phase architecture performs the transform phase simultaneously with one of the other phases (data input or data output). The 3-phase architecture, called also a streaming architecture, provides simultaneous execution of the three mentioned phases. (The processor blocks presented in *figures 7-9* are for the 1-phase radix-2/radix-4 architectures. 2-phase architectures require one more DM/QM per PB and 3-phase architectures require two more DMs/QMs per PB).

Of these three types of architectures 1-and 3-phase radix-2 and radix-4 architectures are implemented. The implementations are made on Xilinx FPGA (Spartan3, Virtex 5) for different values of the input parameters. The physical realization, as well as the testing in a real mode is performed for radix-2, 4-processor architecture for FFT on 256 points, based on a development board containing a FPGA Spartan 3E series of Xilinx company. For the purposes of development ISE Webpack 9.1/10.1 products of the same company are used.

The resource utilization is a linear function of all input parameters.

The system latency (L) and throughput (T) are as follows:

(12) $L = \log_R N\,(N/(PR) + K - 1)\, t_{clk}$,

$T = N L^{-1}$,

where

P is number of processors,

K is the pipeline length – 7 cycles for radix-2 case and 8 cycles for radix-4 case;

R is the radix.

The described systems provide the following advantages:

- Parametric generation and utilization in a wide operational range of input parameters.
- Utilization of distributed shared dual-port memories, as well as their connection with the processor elements according to the perfect shuffle rule, solving efficiently two basic problems - switching of the intermediate data between processors, and memory conflicts.
- Uniform control for the different PBs – realization of SIMD architecture.
- Theoretically, these architectures allow unlimited scalability depending in practice only on the FPGA capabilities.
- The system performance is a linear function of the number of PB.
- The optimal resource utilization is a linear function of all input parameters.
- The implementation of a block-floating point provides efficient strategy for treating the overflow problem.

The comparison between the two (complex FFT and FHT/RFFT/FCT) architectures (with respect to performance and efficiency, and resource utilization), shows that an N-point FFT architecture in practice is equivalent to a 2N-point FHT/RFFT/FCT architecture.

Summary

Natural architectures are such architectures, which are derived from the signal graph (flow of data) of the corresponding algorithm. That is why they are considered to be the most appropriate architectures for parallel realization of this algorithm. For the *Fast Fourier Transform* algorithm (FFT) two types of natural architectures are known – the *direct* and the *indirect hypercube*. The direct hypercube has been investigated and analyzed a long time ago. The development of the concept of “Indirect Hypercube”, although quite old, is too difficult, controversal and still unfinished.

The transpose type algorithms are important parallel FFT algorithms which dominate in the two main trends of parallel implementation of parallel FFT algorithms. The first trend, developed for the goals and tasks of computational mathematics, is associated with the development of modern MIMD computer architectures with distributed memory. The second trend is connected with the development of embedded fast FFT processors necessary for fast real time applications.

Fast Hartley transform (FHT)/Real-valued Fast Fourier transform (RFFT) algorithms are important Fourier-related transforms, because they lower twice the operational and memory requirements when input data is real-valued. These types of algorithms, however, have irregular computational structure, which makes their parallel implementation a very difficult task.

Based on the results achieved so far, this paper presents a further development of the concept Indirect Hypercube as a natural architecture for FFT/RFFT/FHT/FCT parallel transpose type algorithms.

A method for dataflow presentation and analysis of a parallel radix-2 FFT algorithm is proposed, based on multistage interconnection networks (Omega network). Two types of identifiers (group and local) are derived from the global identifier (index) and their properties are analyzed. Based on the main properties of these two identifiers, parametrical synthesis of an indirect binary hypercube is performed.

The generalization with respect to granularity/radix of the above mentioned method is performed. It is based on multistage interconnection networks (*Omega-similar* network) and allows dataflow presentation and analysis of the *transpose* type algorithms with all possible values of granularity/radix. The properties of the group and local identifiers are generalized, as well as the parametrical synthesis of the indirect hypercube on the basis of the corresponding *Omega-similar* networks.

Generalization with respect to RFFT/FHT/FCT of the above mentioned method is performed. Converting the standard FFT transpose type algorithms allows creation, dataflow presentation and analysis of the relevant RFFT/FHT/FCT *transpose* type algorithms.

Two types of SIMD array FFT/RFFT/FHT architectures are described (radix-2 and radix-4), based on the corresponding indirect hypercube.

These two types of architectures are implemented as fast FFT processors for real time applications. The performance estimation is carried out, as well as estimation of the resource utilization.

References

1. [Hartley-1] Hartley, R. V. L. A More Symmetrical Fourier Analysis Applied to Transmission Problems. Proc. IRE 30, 1942, 144-150.

2. [Danielson-1] Danielson, G. C. and C. Lanczos. Some Improvements in Practical Fourier Analysis and Their Application to X-ray Scattering From Liquids. J. Franklin Inst., 233, April 1942, 365-380, 435-452.

3. [Cooley-1] Cooley, J. W., J. W. Tukey. “An Algorithm for the Machine Calculation of Complexes Fourier Series. – *Math. Comput.*, 19, 1965, No. 90.

4. [Singleton-1] Singleton, C. A. Method for Computing the Fast Fourier Transform with Auxiliary Memory and Limited High-Speed Storage. – *IEEE Transactions on Audio and Electroacoustics*, AU-15, June 1967, 91-98.

5. [Bergland-1] Bergland, Glenn D. **A Fast Fourier Transform Algorithm for Real-valued Series**. – *Communications of the ACM*, 11, Oct. 1968, 10, 703-710.
6. [Stone-1] Stone, H. S. Parallel Processing with the Perfect Shuffle. – *IEEE Trans. Computers*, C-20, 1971, 153-161.
7. [Ahmed-1] Ahmed, N., T. Natarajan and K. R. Rao. Discrete Cosine Transform. – *IEEE Transactions on Computers*, Jan. 1974, 90-93.
8. [Pease-1] Pease, M. C. The Indirect Binary n-cube Microprocessor Array. – *IEEE Trans. on Computers*, May 1977.
9. [Bracewell-1] Bracewell, R. N. Discrete Hartley Transform. – *J. Opt. Soc. Am.,* 73, 1983, 12, 1832-1835.
10. [Bracewell-2] Bracewell, R. N. The Fast Hartley Transform. – *Proc. IEEE*, 72, 1984, 8, 1010–1018.
11. [Ulman-1] Ulman, Ronald F. An Algorithm for Fast Hartley Transform. Technical Report, Stanford University, 1984.
12. [Bhuyan-1] Bhuyan, L. Interconnection Networks for Parallel and Distributed Processing. – *Computer*, June 1987, 9-12.
13. [Sorensen-1] Sorensen, H. V., D. L. Jones, M. T. Heideman, C. S. Burrus. Real-valued Fast Fourier Transform Algorithms. – *IEEE Trans. Acoust. Speech Sig. Processing,* 35, 1987, 849–863.
14. [Marchesi-1] Marchesi, M., G. Orlandi, F. Piazza. A Systolic Circuit for Fast Hartley Transform. IEEE International Symposium on Circuits and Systems, 7-9 June 1988, 2685-2688.
15. [Rao-1] Rao, K., P. Yip. Discrete Cosine Transform: Algorithms, Advantages, Applications. Boston, Academic Press, ISBN 0-12-580203-X, 1990.
16. [Zapata-1] Zapata, E. L., F. Arguello, F. F. Rivera, J. D. Bruguera. Multidimensional Fast Hartley Transform onto SIMD Hypercubes. – *Microprocessing and Microprogramming*, 29, September 1990, 2, 121-134.
17. [Chang-1] Chang, L.-W., S.-W. Lee. Systolic Arrays for the Discrete Hartley Transform. – *IEEE Transactions on Signal Processing***,** 39, Nov. 1991, 11, 2411-2418.
18. [Zapata-2] Zapata, E. L., F. Arguello. A VLSI Constant Geometry Architecture for the Fast Hartley and Fourier Transforms. – *IEEE Transactions on Parallel and Distributed Systems*, 3, Jan. 1992, 1, 58-70.
19. [Liu-1] Liu, K. J. R., C.-T. Chiu. Unified Parallel Lattice Structures for Time-recursive Discrete Cosine/Sine/Hartley Transforms. – *IEEE Transactions on Signal Processing,* Mar. 1993, 1357-1377.
20. [Arguello-1] Arguello, F., J. D. Bruguera, R. Doallo, E. L. Zapata. Parallel Architecture for Fast Transforms with Trigonometric Kernel. – *IEEE Transactions on Parallel and Distributed Systems*, 5, Oct. 1994, 10, 1091–1099.
21. [Uniyal-1] Uniyal, P. R. Transforming Real-valued Sequences: Fast Fourier Versus Fast Hartley Transform Algorithms. – *IEEE Transactions on Signal Processing*, 42, Nov. 1994, 11, 3249-3254.
22. [Mazzeo-1] Mazzeo, A. and U. Villano. Parallel 1D-FFT Computation on Constant-Valance Multicomputers. – *Software – Practice and Experience*, 25, June 1995, 6, 681-704.
23. [Zhenyu Liu-1] Zhenyu, Liu, Yang Song, Takeshi Ikenaga, Satoshi Goto. A VLSI Array Processing Oriented Fast Fourier Transform Algorithm and Hardware Implementation. GLSVLSI’05, 17–19 April 2005, Chicago, Illinois, USA.
24. [Jones-1] Jones, K. J. Design and Parallel Computation of Regularised Fast Hartley Transform. – *Vision, Image and Signal Processing – IEEE Proceedings*, 153, 9 Feb. 2006, 1, 70-78.
25. [Phil-1] Philipov, Ph., V. Lazarov, Z. Zlatev, M. Ivanova. A Parallel Architecture for Radix-2 Fast Fourier Transform. IEEE John Vincent Atanasoff 2006 International Symposium on Modern Computing, Sofia, 2006, 229-234.
26. [Phil-2] Philipov, Philip, Ilian Costov, Vladimir. Lazarov, Zlaty Zlatev, Milena Ivanova. Implementation of a Parallel Architecture for Radix-2 Fast Fourier Transform. – *Information Technologies and Control*, ISSN 1312–2622, 2008, No. 2, 12-16.
27. [Phil-3] Philipov, Ph. Investigation of the Indirect Hypercube as a Natural Architecture for Realization of the Fast Fourier Transform Algorithm. PhD Thesis, Sofia, 2010.

Manuscript received on 5.12.2013

__Philip Philipov__ was born in 1952. He graduated the Technical University – Sofia as an engineer in electronics. Since 1993 he is with the Institute for Information and Communication Technologies (ICCT), Bulgarian Academy of Sciences, as a research associate. In 2010 he received PhD degree. Since 2012 he is with EPU Pernik as an assoc. professor in the field of Informatics and computing technique. His main interests are computer architectures, parallel processing and analytical modeling.

Contacts:
Institute of Information and Communication Technologies
Bulgarian Academy of Science, Sofia
e-mail: filip@bas.bg

__Vladimir Lazarov__ was born in 1942. He graduated the Technical University – Sofia as a computer engineer in 1968 and received Ph.D. degree at Sankt Petersburg Electrical Institute in 1974. From 1976 to 1990 he was chief constructor of many computers and devices in the Central Institute for Computing Technique in Sofia. Since 1993 he is with the Institute for Information and Communication Technologies (ICCT), Bulgarian Academy of Sciences – Sofia as a Principal Scientific Officer. Since 1997 he is a guest professor at Technical University – Plovdiv. Since 2011 he is Head of BA Program “Informatics” and MA Program “High Performance Computer Systems” at EPU, Pernik. His main interests are computer architectures, parallel processing, supercomputing and simulation.