Implementation of a Parallel Architecture for Radix-2 Fast Fourier Transform*

Ph. Philipov, I. Costov, V. Lazarov, Z. Zlatev, M. Ivanova

Key Words: *Parallel Fast Fourier Transform; high-performance computer architectures.*

Abstract. *This paper describes the main problems, connected with realization of the Fast Fourier Transform (FFT) algorithm on a radix-2 parallel architecture, suitable for implementation on field programmable gate arrays (FPGA) and based on the perfect shuffle interconnection pattern. Performance estimation and resource utilization analisis are presented. The main advantages and possible applications of the proposed architecture are outlined.*

1. Introduction

The discovery of the Fast Fourier Transform (FFT) algorithm by Cooley and Tukey [1] in 1965 is a key point in the development of digital algorithms and parallel calculations. FFT plays significant role in several important scientific and technical areas as solving of partial differential equations, spectral analysis, digital signal processing, communication, etc.

As we know, parallelism is an intrinsic feature of FFT. Parallel realizations of FFT algorithm have been implemented on almost all high-performance parallel computer architectures – vector computers, systolic arrays, transputer systems, processor meshes with different organization, etc. The main approach of these realizations is the optimal mapping of the algorithm on the concrete architecture. Leading factors in this process are efficiency, performance and scalability [7]. Switching and routing of data between many processors (processor elements – PE), as well as memory conflicts, are the basic problems connected with the FFT parallel realization.

Since the paper of Cooley and Tukey [1] a large number of FFT algorithms were developed. Among them the radix-2, radix-4, split-radix and FHT (Fast Hartley Transform) are the mostly used algorithms for practical applications. Nowadays, modern investigations in parallel FFT algorithms and architectures are concentrated over the solution of two main problems:

- Fast *Poisson* solvers.
- Fast FFT processors for real-time digital applications.

Fast Poisson solvers, based on FFT, are considered to be among the fastest techniks for solution of the *poisson equation* on unidorm grids.

2. Fast FFT Processors

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

Advance in technologies enables development of architectures, optimal for a given class of algorithms, including FFT. Recently there have been dedicated considerable efforts for FPGA-based parallel realizations of algorithms 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 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 arbitrary level of detailing, as well as the parameterization of any type of characteristics.

Generally, there are three main ways of improving the FFT speed, all of which based on the *Transpose* algorithm:

1. Increasing the order of the radix. This approach improves both latency and throughput, but is costly in terms of computational resource required for each processing element.
2. Cascading (pipelining) the processors. Different processors operate over different stages. Cascading improves throughput, but not latency. Interstage memory is necessary.
3. Parallelling the processors, so that a single large FFT is divided into N smaller FFTs. Both latency and throughput are improved with this arrangement. Drawback is the data transfer between stages.

A different approach is applied by Zhenyu Liu et. al. [11], proposing an *Array processing* architecture for parallel FFT implementation. Main advantage of this architecture is the elimination of inter stage data transfer, so both the system throughput and latency are improved.

It is known that the indirect binary hypercube [4, 8] architecture *seams to be one of the most appropriate architec-*

* This work is supported and performed in the frame of Project Nr. DO 02-115 of the Bulgarian National Science Fund(NSF)

tures for FFT parallel implementation.

In 1995 Macceo et. al. [8] mapped the butterfly diagram on an indirect binary hypercube. In the proposed algorithm however, the workload was unevenly distributed among the different processing units, which lead to decrease in efficiency.

The *Array processing* architecture, mentioned above, owes its notable characteristics to the fact that it contains in a *hidden* form an indirect binary hypercube. This hidden form however hampers significantly the formulation of general rules, concerning memory addressing and operation, as well as organization and addressing of the tables containing the twiddle factors. These rules are necessary for the formulation of a general algorithm for implementation of FFT on this architecture.

This paper represents continuation of the research, done in [14] and [15]. There, on the basis of detailed analysis of the FFT flow of data performed with the help of multistage interconnection networks (MIN, Omega network) is proposed a parallel architecture for 1-D, radix-2 FFT with indirect hypercube topology and suitable for FPGA implementation. Formulated are also general rules concerning memory addressing and operation, as well as organization and addressing of the tables containing the twiddle factors (CLUT).

3. The Architecture

3.1. General

FFT (IFFT) are algorithms, implementing the direct and inverse discrete Fourier transforms (DFT, IDFT) which are defined as follows [2]:

$$(1) \quad f(k) = \sum_{n=0}^{N-1} A(n) W^{kn};$$

$$A(n) = \frac{1}{N} \sum_{k=0}^{N-1} f(k) W^{-kn}$$

where W is the main N-th root of the one, $N=2^m$.

Let $N = 2^n$, $n = n_1 + n_2$, $1 < n_1 < n - 1$.

In this paper, we examine the FPGA implementation of this architecture with the following features [15]:

- A set (2^{n1}) of processor elements (PE) is connected to a set of dual-port memories (DM) with direct access for reading and/or writing, performing N-point FFT.
- Every PE is connected to one DM for reading through the two ports, thus forming a processor block (PB), and with two DMs (from the same or another PB), for writing through one of the DM ports.
- PBs are connected in perfect shuffle manner, thus forming an indirect hypercube topology.
- Interconnection between PBs - perfect shuffle interconnection pattern - is based on coincidence of PB input and output identifiers.
- PB input identifier (a binary number) includes the input number (0 or 1) as most significant digit, followed by the PB number.
- PB output identifier (a binary number) includes the PB number as most significant part, followed by the output number (0 or 1) as least significant digit.
- Every PE contains a look-up table memory (CLUT) for the necessary coefficients (twiddle factors), the necessary register structures and arithmetic and control units.
- Addressing patterns of DMs and CLUTs are derived from the scalar case (1 PE), and are based on the left cyclic rotation of the local indentifier [15].
- Tightly coupled system, operating synchronously, controlled by a main control unit (MCU).
- Uniform control for the different PE/DM – realization of SIMD paradigm.

On *figure 1* is presented the block diagram of architecture, consisting of four PBs.

3.2. Preliminary Notes

Different types of organization can be considered for this architecture (in fact class of architectures). Possible are 1-phase, 2-phase and 3-phase architectures.

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, further referred as streaming architecture, provides simultaneous execution of the three mentioned phases.

The most popular strategies for treating the overflow problem are:

- Full-precision unscaled arithmetic.
- Scaled fixed-point, where the user provides the scaling schedule.
- Block floating-point.

Full-precision unscaled arithmetic is applicable for small number of points (N) because the size of output and intermediate busses grows very rapidly with the growth of N. The width of the output will be the input width + number of stages + 1 [16].

Scaled fixed point has the two following disadvantages:

- Analysis of input data for prescaling before each stage (or couple of stages).
- Possibility for occurrence of overflow or loss of accuracy, when prescaling is not relevant.

Block floating point provides automatic scaling every stage, but sometimes, for parallel structures, is very arduous for implementation.

For the architecture under discussion, the block floating point alternative is possible and not so difficult for implementation, so it is chosed to be the strategy to deal with the overflow problem.

3.3. General Structure and Topology.

The generated architecture includes the following units:

1. Processor blocks.
2. FFT control unit (FFTCU).
3. Data flow control unit (DFCU).
4. Input/output block (IOB).
5. Block exponent control unit (BECU).

The basic building block of this architecture is the processor block (PB). PB for the 1-phase architecture includes one

processor element (PE) operating as butterfly unit and two DMs. For a given stage of the FFT transform one of these two DMs is used by PE as source of data for the current butterfly operations and the other DM is used for storing the results of the current butterfly operations of other PBs (two), which are connected with the given PB. DMs alternatively change their role on stage basis. For the multi-phase architectures, every new phase requires one DM more per PB. On *figure 2* is presented the PB block diagram for 1-phase architecture.

PE is realized as a 7-stage linear pipeline. It computes radix-2 butterflies and yields a result every cycle. Input data, as well as intermediate results are presented in fixed point, 2's complement format. Twiddle factors (unique for different PEs) are held in CLUTs. From coefficients viewpoint, the stages are divided into two classes: low stages and high stages. Low stages are the first n_1 stages. 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 stage. For the high stages every PE needs totally 2^{n2-1} coefficient pairs. The set of coefficients for the high stages of given PE includes all the coefficients, whose arguments have, as most significant digits of their binary presentation, the bit reversed number of PB.

Two variants of complex multiplication are possible – standard (4 multiplications/2 additions) and nonstandard (3 multiplications/5 additions). DIT (decimation-in-time) or DIF (decimation-in-frequency) methods can be used for FFT computation (forward and inverse).

Very suitable for DMs are the dual-port block memories, provided by Xilinx in their FPGA.

The Input/Output block (IOB) performs I/O functions – the input and output of frames of points. Input is in natural order. Output can be in natural or bit-reversed order.

FFT control unit provides the necessary control information for PBs – synchronization of the arithmetic pipelines, addresses for DMs and CLUTs, etc.

DM addressing is in accordance with the *perfect shuffle* method. Generated are four address sequences for DMs – two read address and two write address sequences. The algorithms for generation of these address sequences are similar to the respective algorithms in the scalar case (one PB) –left/right cyclic rotation of store address sequences for the case of natural/bit reversed input ordering of initial data. 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.

FFT control is performed on SIMD paradigm – the control information (PE pipeline control, DM addresses and CLUT addresses) is the same for all PBs.

DFCU controls and synchronizes the data flows for the different phases – data input, transform and data output.

4. System Generation

4.1. General

The main parameters, specifying the system generation and operational characteristics of PE, DM and CLUT are:

- n – exponent, specifying the maximum number of points in one dimension ($N = 2^n$). This parameter specifies the total memory capacity of DMs.
- n_1 – exponent, specifying the number of PBs ($NPB = 2^{n1}$).
- Data width/twiddle factors width – typically 16÷24 fixed point, 2's complement format;
- Output – natural/bit reversed order.
- Number of phases – 1, 2 or 3-phase architecture.

VHDL possesses excellent features for parameterized generation – the *generic* option, by means of which parameters can be specified and passed to lower layers of the architecture, as well as the *generate* operator, which enables parametric generation of structures.

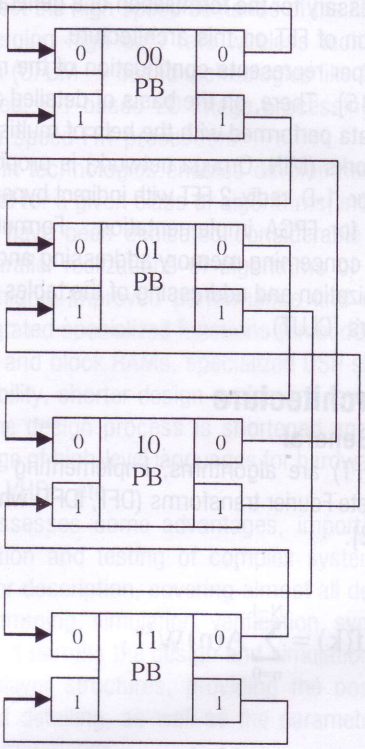


Figure 1. A 4PB architecture

The *generate* operator is used for generation of the necessary number of PB (NPB). Data busses of two types are also generated – PEBUS and PBBUS. PEBUSes are the internal busses for PBs. PBBUSes are used for interconnection between PBs. There exist totally 2.NPB PBBUSes. Every PB uses 4 PBBUSes – 2 for data input and 2 for data output. Assignment of PBBUSes to PBs and interconnection between PBs according to the perfect shuffle rule is accomplished as follows:

1. To the input ports (0 and 1) of PB(i) are attached PBBUS(i) and PBBUS(i + NPB).

2. To the output ports (0 and 1) of PB(i) are attached PBBUS(2*i) and PBBUS(2*i +1).

Generation of unique private CLUTs is accomplished with the help of the PB identifier, which is passed as parameter by means of the *generic* function. All constants, data types and functions, necessary for the system generation are included in a special VHDL package, thus improving the transparency and readability of the program.

All phases of implementation (synthesis, simulation, mapping, placing, routing and programming) have been realized with

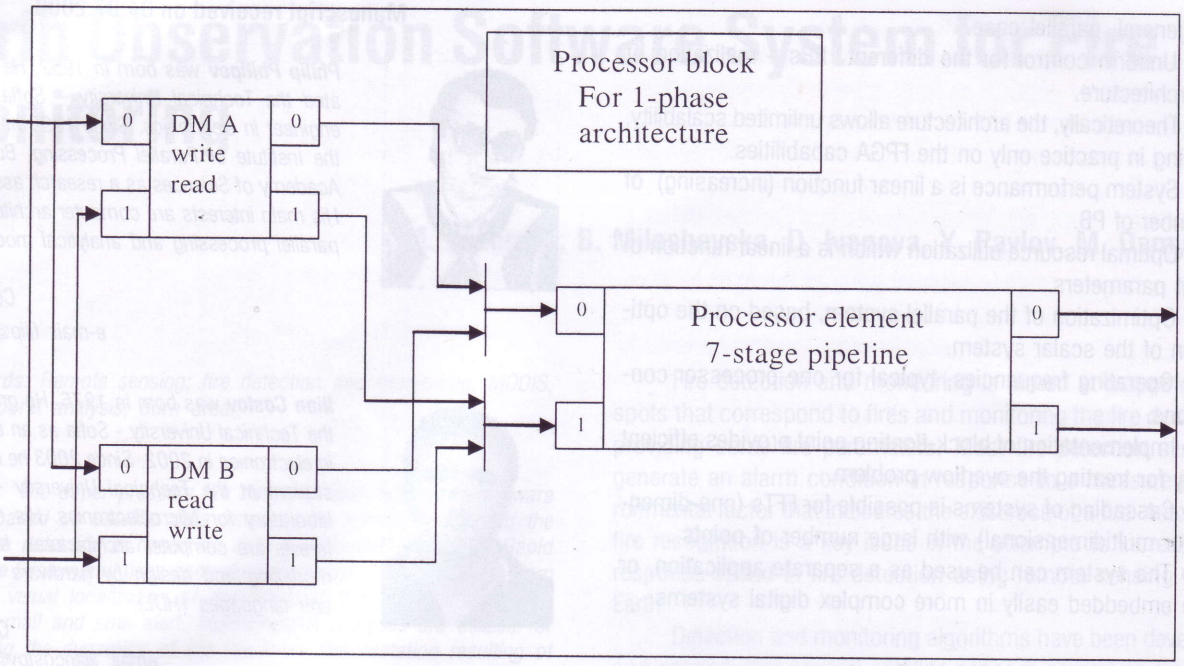


Figure 2. Processor block

the help of the free product of Xilinx Corporation – Webpack ISE–-9.1i.

5. Implementation Results

All three types (1, 2 and 3-phase) of architectures have been implemented on Xilinx FPGA (Spartan3, Virtex 2,4 and 5) for different input parameters.

5.1. Performance

System latency (L) and throughput (T) are as follows:

$L = n.(2^{n2-1} + P_l).t_{clk}$;

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

P_l is the pipeline length – 7 cycles, t_{clk} is the system clock.

For optimal utilization of the proposed hardware, some relations between I/O speed and transform speed should be regarded. Let T_i, T_o and T_t be the input, output and transform time in cycles. Assuming that $T_i = T_o = N.t_{clk}$, we see that, for the 3-phase architecture, we have no gain if we make $Tt << N.t_{clk}$. This results in the following relation:

$Pmax \approx n/2$, where

Pmax is the maximum number of PEs, for which we have $Tt \geq Ti$.

One way to avoid this restriction is to provide multichannel input/output [16].

When necessary, for large one-dimensional or multidimensional FFTs, such systems can be cascaded, thus increasing the throughput, respectively performance of the whole system.

All implementations show operational frequencies, which are typical for the one-processor configuration, implemented on the respective device.

5.2. Resource Utilization

Resource utilization is a linear function of all input parameters. In details, we have the following dependences:

1. FFTCU, DFCU and IOB resources in practice are very slightly dependent from input parameters (with the exception of bus width for IOB).

2. Total DM capacity is a linear function of N and the phase type (1, 2 or 3-phase architecture), and does not depend on NPB. Configuring DMs as quad-port memories (QM) we can have some gain in DMs (1, 2 or 3 QMs per PB will be necessary for the 1-, 2- and 3-phase architecture respectively). Configuring a DM as a QM is at the expense of lowering its operational frequency twice, which results in lowering the operational frequency of the whole system.

3. Total capacity of CLUTs in practice is also a linear function of N and is very slightly dependent on NPB.

4. BECU resources are a linear function of NPB.

5. Other recourses of PB – arithmetic units, data busses, etc. – are linear functions of the bus width and do not depend on NPB.

We can make the conclusion, that, in order to realize an optimal version of this architecture it is sufficient to get an optimized version of 1-processor architecture (which is already not a problem) and make it a multiprocessor architecture.

6. Conclusions

The described system provides the following advantages:

- Parametric generation and utilization in a wide operational range of input parameters.
- The utilization of distributed shared dual-port memories, as well as their connection with the processor elements according to the perfect shuffle rule, solves efficiently the two basic problems - switching of intermediate data between processors, and memory conflicts.
- The butterfly/perfect shuffle methods and algorithms for DM addressing and CLUT generation and addressing, which are valid for the scalar case, are also valid and easily implemented

in the general, parallel case.

- Uniform control for the different PBs – realization of SIMD architecture.
- Theoretically, the architecture allows unlimited scalability, depending in practice only on the FPGA capabilities.
- System performance is a linear function (increasing) of the number of PB.
- Optimal resource utilization which is a linear function of all input parameters.
- Optimization of the parallel system, based on the optimization of the scalar system.
- Operating frequencies, typical for one-processor configurations.
- Implementation of block-floating point provides efficient strategy for treating the overflow problem.
- Cascading of systems is possible for FFTs (one-dimensional or multidimensional) with large number of points.
- The system can be used as a separate application, or can be embedded easily in more complex digital systems.

References

1. Cooley, J, W., J. W. Tukey. An Algorithm for the Machine Calculation of Complexes Fourier Series.- Math. Comput., 19, 1965, № 90.
2. Marchouk, G. I. Methods of Computational Mathematics. Moscow, Ed. Science, 1980, 224-228 (in Russian).
3. Stone, H. S. Parallel Processing with the Perfect Shuffle. –*IEEE Trans. Computers*, C-20, 1971, 153-161.
4. Pease, M. C. The Indirect Binary n-cube Microprocessor Array.– *IEEE Trans. on Computers*, May 1977.
5. Swarztrauber, Paul N. Multiprocessor FFTs. *Parallel Computin* 5,1987, 197-210.
6. Bhuyan, L. Interconnection Networks for Parallel and Distributed Processing. – *Computer*, June 1987, 9-12.
7. Gupta, A., and V. Kumar. The Scalability of FFT on Parallel Computers.– *IEEE Transactions on Parallel and Distributed Systems*, 4 (8), August 1993.
8. Mazzeo, A., and U. Villano. Parallel 1D-FFT Computation on Constant-Valance Multicomputers. –*Software - Practice and Experience*, 25 (6), 1995.
9. Mermer, Coskun, Donglok Kim and Yongmin Kim. Efficient 2D FFT Implementation on Mediaprocessors. –*Parallel Computing*, 6, June 2003, 691 – 709.
10. Pilz, N., B. Lerner and K. Adamson. Parallel FFT Implementations on Fixed-Point DSP-Cores with Subword-Parallelism. ISSC 2005, Dublin 2005, 338-345.
11. Liu, Zhenyu, Yang Song, Takeshi Ikenaga, Satoshi Goto. A VLSI Array Processing Oriented Fast Fourier Transform Algorithm and Hardware Implementation, GLSVLSI'05, April 17 – 19, 2005, Chicago, Illinois, USA.
12. Xu, Xizhen and Sotirios G. Ziavras. A Coarse-Grain Hierarchical Technique for 2-Dimensional FFT on Configurable Parallel Computers.– *Ieice Trans. Inf. & Syst.,E89-D*, Feb. 2006, No. 2.
13. Eleftheriou, M. et al. Scalable Framework For 3d FFTs On The Blue Gene/L Supercomputer: Implementation and Early Performance Measurements.– *IBM J. Res. & Dev*, 49 March/May 2005, No. 2/3.
14. Philipov, Ph., V. Lazarov, Z. Zlatev, M. Ivanova. Some Problems of Fast Fourier Transform Parallel Implementation. International Conference Automatics and Informatics '05, Sofia, 2005, 87-90.
15. 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.
16. Xilinx Corporation. Fast Fourier Transform, 4.1, DS260 April 2, 2007.

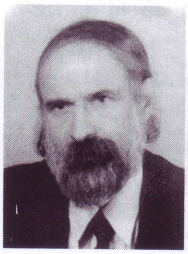
Manuscript received on 04.07.2008

Philip Philipov *was born in 1952. He graduated the Technical University – Sofia as an engineer in electronics. Since 1993 he is with the Institute of Parallel Processing, Bulgarian Academy of Sciences as a research associate. His main interests are computer architectures, parallel processing and analytical modeling.*

Contacts:
e-mail: filip@bas.bg

Ilian Costov *was born in 1976. He graduated the Technical University - Sofia as an engineer in electronics in 2002. Since 2003 he is a PhD student at the Technical University – Sofia, laboratory for Microelectronics. His main interests are computer architectures, hardware modelling and design by hardware description languages (HDL).*

Contacts:
email: iliancostov@mail.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 of Parallel Processing, Bulgarian Academy of Sciences as a Principal Scientific Officer, head of Department and Vice Director. Since 1997 he is a guest professor at Technical University – Plovdiv. His main interests are computer architectures, parallel processing, supercomputing and simulation.*

Contacts:
e-mail: lazarov@bas.bg

Zlatko Zlatev *was born in 1947. He graduated the Technical University - Sofia as a computer engineer. Since 1993 he is with the Institute of Parallel Processing, Bulgarian Academy of Sciences as an Associated Professor. His main interests are computer architectures, parallel processing and performance analysis.*

Contacts:
e-mail: zlaty@bas.bg

Milena Ivanova *was born in 1944. She graduated the Sofia University- Sofia as a mathematician. Since 1993 she is with the Institute of Parallel Processing, Bulgarian Academy of Sciences as an Associated Professor. Her main interests are computer architectures, parallel processing and simulation modeling.*

Contacts:
e-mail: milen@bas.bg