

# **Parallel FPGA Processor Card for Distributed Information Processing.**

**Sergey Aryashev, Sergey Bobkov,  
Evgueni Sidorov, Ilya Yudin**

## **I. Introduction.**

Progress in information technology has led to an ever increasing demand for more advanced data processing techniques and devices. Typically very fast and complicated computers are required to perform many current and future computational tasks in a useful amount of time. There are two predominant approaches to the designing of computers that satisfy such demands. The first is a hardware implementation of the computational algorithms and the second is the use of a parallel architecture. Both approaches have proven to be quite effective and are widely used in many applications. However, with all their advantages they still have drawbacks. Hardware implemented algorithm systems are usually designed and optimized for specific tasks and a given system can only be used in a narrow class of applications. If conditions or requirements change, a new hardware design has to be implemented. The parallel computing approach requires very sophisticated methods of distributing data and processes between different processing and storage units in order to achieve adequate performances. These problems have been and continue to be the subject of much ongoing research.

In this paper a Parallel Field Programmable Gate Array (FPGA) Processor Card (PFPC-1V) for distributed information processing is presented. This device combines both parallelism and hardware implementation of algorithms to provide high computation speeds for a number of varied applications. And to eliminate the drawback of a fixed non-universal architecture inherent in hardware implemented computational algorithms, FPGA's are used. The use of FPGA chips allows the local architectures of processing units to be easily changed so computational algorithms of different kinds can be implemented in hardware just by an on-line reconfiguration of the appropriate FPGA. The PFPC-1V has been found useful in a wide variety of fields such as image processing, pattern recognition, artificial neural networks modeling, etc.

## **I. The Hardware.**

### **II.1. The Structure and a Brief Description.**

The current PFPC-1V was developed as VME-bus card and all its basic processing units are built using the Altera FLEX 10K family of FPGA chips. The basic implementation is designed as a hardware accelerator and must be used as a slave on VME bus with a conventional general purpose processor as a master (host-machine). Architecturally, a PCI bus version would be similar and re-implementation on a PCI bus would be straight forward. The current PFPC-1V works at frequency of 33 MHz.

Figure 1 shows a block diagram of the system architecture. The system consists of the following units:

- Control Unit (CU);
- six Base Processing Units (BPU1-BPU6);
- Exchange Bus Controller (E-bus Controller);
- VME bus Controller;
- two SRAM arrays (SRAM1,SRAM2);
- high-speed serial receivers/transmitters.

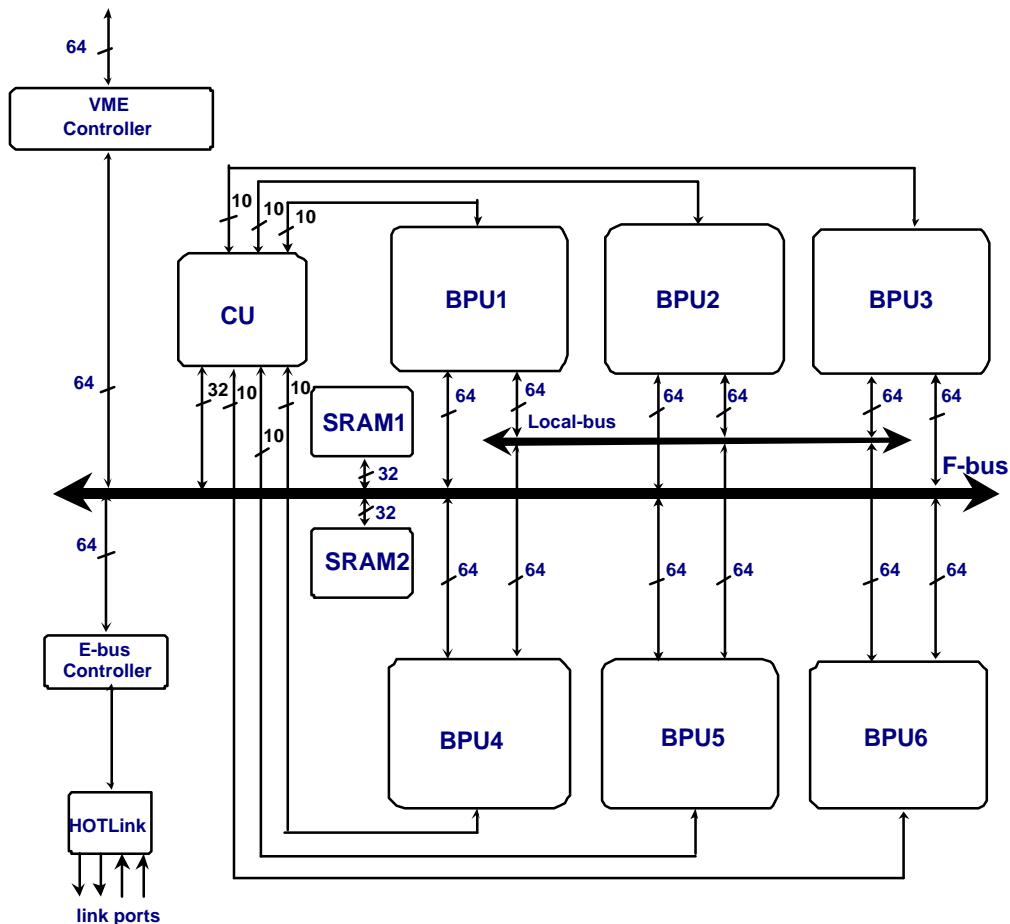


Fig.1. The Parallel FPGA Processor Card structure.

The Control Unit controls the BPUs and manages the data flows on the card. It can be considered as a RISC processor whose processor structure and instruction set can vary depending on the particular task requirements. The actual operations on the data are performed by the BPUs, whose architectures are hardware implementations of the corresponding algorithms. In the BPUs, different kinds of adders, multipliers, decoders, ALUs, table functions and so on can be implemented, thus providing optimal realizations of the data processing algorithms. Since the CU and BPUs are implemented in Altera FLEX 10K family FPGA chips, their structure can be changed simply by loading new configuration files over the VME.

Local memory consists of two SRAM arrays made up of eight 512Kbyte SRAM chips for a total of 2 Mbyte of memory per array. Each array is arranged as 512K 32-bit words. Both arrays are connected with the CU and the VME Controller

through separate address and control lines and can operate independently from each other.

Two or more Parallel FPGA Processor Cards or any other cards with the appropriate interfaces can be interconnected through the serial receiver/transmitter HOTLink™ channel, by Cypress Corp. The E-bus Controller manages the data transfers through the channel. It is also implemented using an Altera FLEX 10K FPGA and consists of four FIFO pipelines and a number of data and control registers. Interfaces with the host-machine are provided by a VME Controller through the VME bus.

For the system programmer the Parallel FPGA Processor Card can be considered a RISC processor (the Control Unit) with six vector processors (the BPUs), which execute MIMD (Multiple Instructions, Multiple Data) instructions (see Fig. 2). All the BPUs and the CU can operate in parallel, the BPUs are connected with each other through a 64-bit bus, with the CU through multiple 10-bit buses and with the local SRAM arrays through a 64-bit bus. A rich set of data and memory buses, interprocessor interconnections, and hardware resources support both parallel operations and direct hardware implementation algorithms. Which in turn provides very high degree of data processing performance.

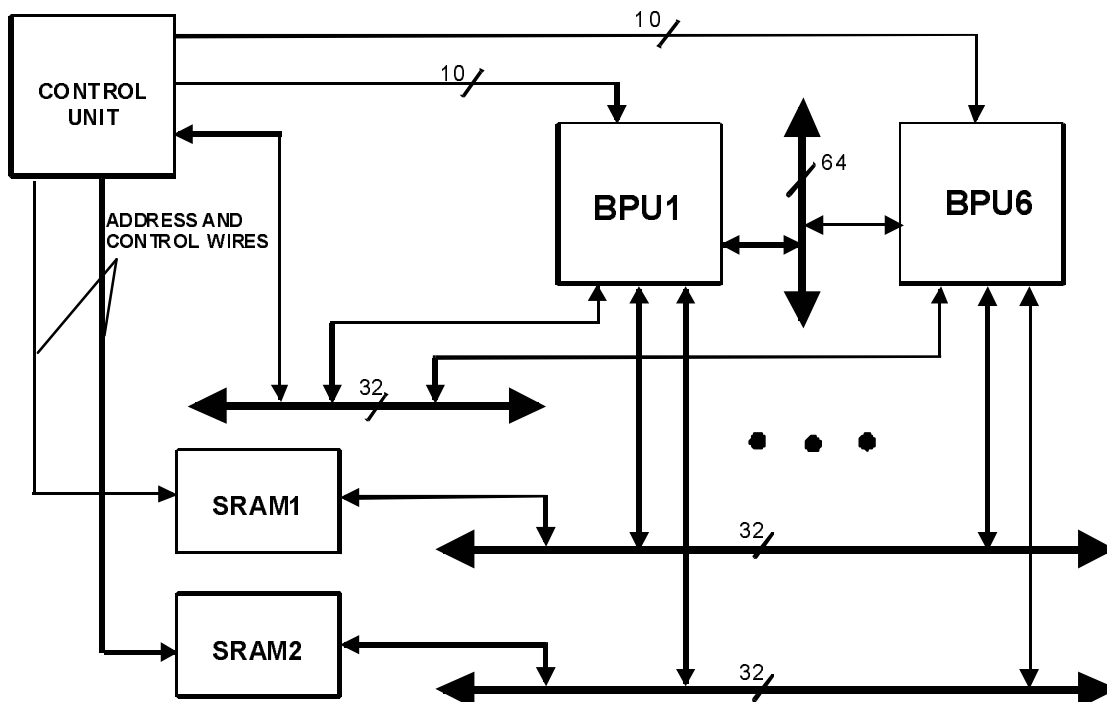


Fig.2. Simplified structure of the Parallel FPGA Processor Card.

## II.2. The Control Unit.

The Control Unit structure is shown in Figure 3. The CU consists of the following logical modules:

- Internal instructions memory (IIM);
- Instruction decoder;
- Data and constants registers;
- Address registers;
- BPU's registers;
- Control unit and status register (CSR);
- ALU;
- Program counter and stack pointer;
- Local SRAM controller;

The active program is stored in the Internal Instructions Memory (IIM), which is implemented in the FLEX 10k Embedded Array Blocks (EAB). Currently the IIM is arranged as 512 32-bit words. The IIM size can be easily increased up to 2048 32-bit words simply by using more powerful chips. The IIM is loaded when the FPGA is configured, or it can be loaded via the VME bus on-line.

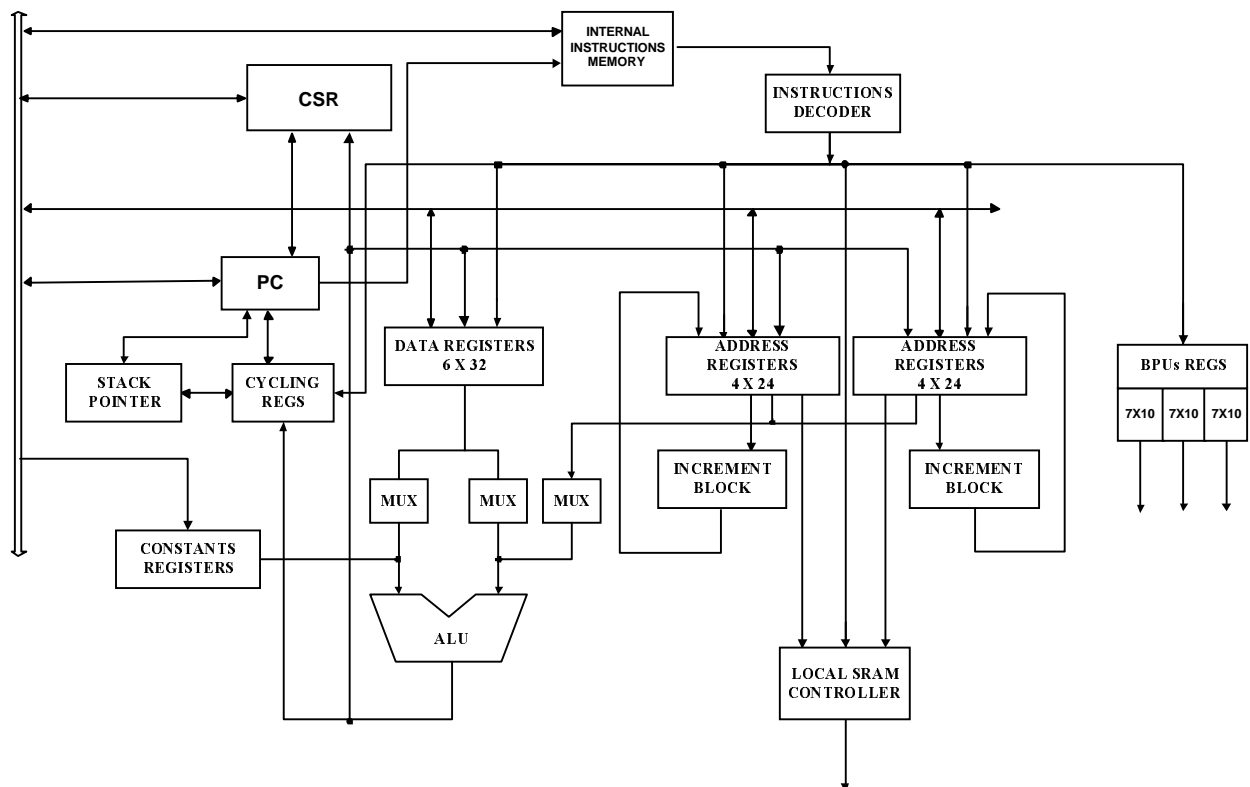


Fig. 3. The Control Unit.

The IIM can be considered a set of subprograms (micro-codes), each of which implements a particular computational algorithm. For example, the first micro-code stored in the 0<sup>th</sup> address is for vector by vector scalar multiplication. The second micro-code, in the 20<sup>th</sup> address, is for matrix transposition. The third micro-code, in

the 60<sup>th</sup> address, is for Hopfield neural network learning. The changeable parameters of micro-codes - sizes of matrices and vectors, numbers of cycles and so on, are stored in the corresponding registers which are loaded from the host-machine through the VME bus before the program starts. Any instruction in the CU processor is executed in three clock ticks. The three-level pipeline allows for a new command to be uploaded every clock tick.

The CU processor has eight 24-bit address registers RA0-RA7, six 32-bit data registers RD2-RD7, eight 8-bit constant registers, twenty one 10-bit BPUs registers and fast branch registers. The CSR register can be considered as the least significant data register RD0. The most significant bits of RD1 register are used as a cycling counter for fast branch instructions and the least significant bits are for cycling the MIMD instructions sent to the BPUs. The RD2-RD7 data registers are used as general purpose registers.

The BPUs registers are used to store the MIMD instructions the CU sends on the 10-bit buses connecting the CU and BPUs. All the BPUs registers are divided into three groups - seven registers for every two BPUs. Only one of seven instructions for a BPU can be available at a moment – the instruction stored in one of the BPU registers - and usually it is good enough for the most applications. The BPUs registers are loaded in the micro-code body and if it is needed, more MIMD instructions can be uploaded.

The address registers are used for handling the addresses of the data stored in the local SRAM arrays. All address registers are divided into two groups with four registers per group. A group corresponds to the particular SRAM array. After execution of the MIMD instruction, any address register can be incremented with the value stored in the special increment registers separately for each group. The need for increment is determined in the CU instruction body for each group individually, i.e. two address registers from different groups can be incremented at the same time. The local SRAM arrays are handled by the local SRAM controller that supports SRAM chips with an access time of 20 ns.

The CU is assigned to manage the data flows and to control the on-card units. For this reason, the CU's ALU has a highly simplified structure. According to the requirements of algorithms implemented to date, the ALU has only had to perform addition, subtraction, shifting and some logical operations. The main CU instruction is for the control of the BPUs. These instructions tell each BPU what register the MIMD instructions should be loaded from, what address registers and what operation mode should be used for the local SRAM arrays, and whether the corresponding address registers should be incremented or not. Each instruction can be repeated for the required number of times according to the value stored at the least significant bits of RD1. This eliminates the losses on branches in the cycles when the MIMD instructions should be performed more than once.

To implement a particular computational algorithm, the specific CU instruction set that allows for the best performance is determined for each task. Then in a hardware design language (HDL), such as Verilog or VHDL, the appropriate CU architecture is described and the corresponding FPGA chip is configured with the architecture supporting the instruction set required. Thus one gets both a specialized computational system optimally adapted to the particular task requirements and a system which can be easily reprogrammed for a new or different task.

## II.3. The Base Processing Units.

The BPUs are assigned to perform elementary operations such as addition, subtraction, multiplication, and other kinds of arithmetic, logical and table functions. A BPU consists of register files, internal SRAM of 2 Kbyte size, logical and arithmetical units. Such resources make it possible to implement, for example, various vector and matrix functions that are basic in image processing or artificial neural networks modeling. A generalized structure of BPU is shown in Figure 4. The particular architecture is determined by the particular task requirements. A BPU executes the instruction that comes from the CU and the data are taken either from the internal storage units or from the local SRAM arrays, or in some cases from other BPUs. That allows for the distributed information processing to be arranged.

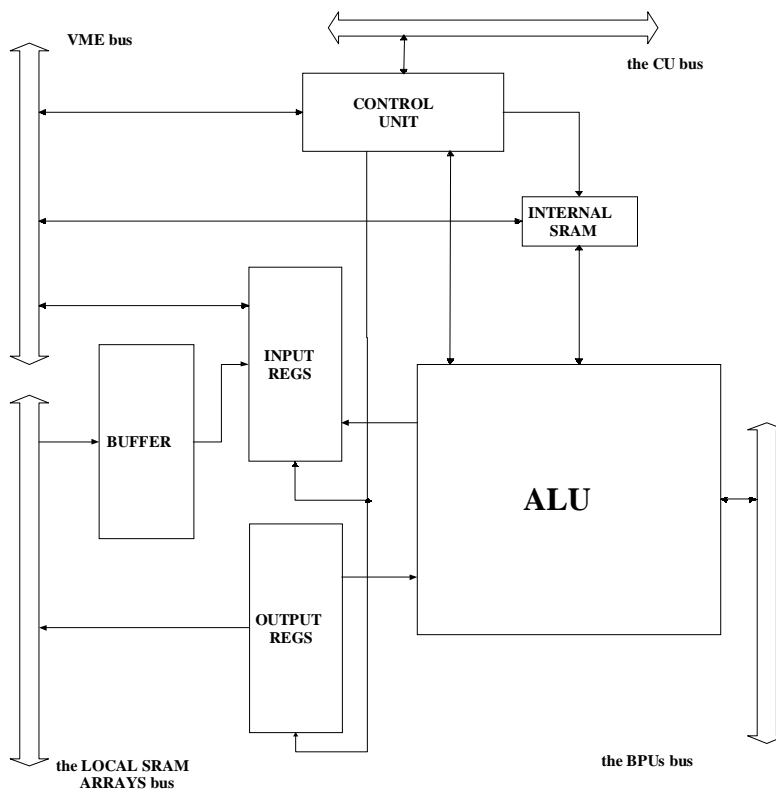


Fig. 4. A generalized BPU structure.

## III. The Software.

The supporting software development is an essential part of designing a computer. In our case we faced the task of developing a flexible programming language with easily changeable sets of commands and instructions, while taking into account the need for some of the CU instructions to provide for the parallel operation of different BPUs and concurrent access to the local SRAM arrays. An assembler-like language was built based on the Dialogue System of Structural Programming (DST). An appropriate compiler was developed that allows for any instruction to be added or deleted easily and the existing instruction codes to be modified in a simple and fast

manner. DST is a FORTH-like language, which has a rich set of facilities for developing low-level debugging and device testing software. In DST a new operator can be added by defining it in terms of previously defined operators. An assembler instruction can be considered as a DST instruction and then an assembler program will at the same time represent a program for DST. Thus the compiling process involves a corresponding DST program execution. As a result a binary code is generated which is then loaded into the IIM of the CU.

#### IV. Application Fields.

The Parallel FPGA Processor Card can be effective in a number of applications. In particular, various algorithms used in image and signal processing can be readily implemented, such as filtering, convolution, Fourier transform, correlation analyses, wavelets, pattern searches, compression/decompression (e.g. MPEG, JPEG), multidimensional analyses, etc. The PFPC-1V card can be used in telecommunications for tasks of coding/decoding and exchange protocols support. In the field of molecular biology the PFPC-1V card is quite effective for DNA and protein sequence alignments: finding similarities using dynamic programming and Smith-Waterman algorithms, searching the databases like GenBank for the closest alignments and other analyses. Another possible use is for artificial neural networks modeling and different kinds of neuron-like nets can be successfully implemented (e.g. Hopfield nets, feed-forward nets). It can be also used for monitoring and control systems as an intelligent controller. A brief review of possible applications is presented in Table 1.

**Table 1.**

<b>Application Field</b>	<b>Examples of Applications</b>	<b>Notes</b>
1. Signal processing	Filtering, Furry transform, convolution, wavelet analyses	See, as analogue, the PAM system, developed by Digital, with various DSP algorithms implemented in FPGA [10]
2. Image processing	Filtering, convolution, contrast enhancing, edges detection	
3. Pattern recognition	Correlation analyses, pattern searches, features extraction	
4. Telecommunications	Compression/decompression (MPEG, JPEG, etc.), protocol interfaces	
5. Neural nets	Hopfield nets, feed-forward nets	
6. Molecular biology	Sequences alignment (dynamic programming, Smith-Waterman algorithm), databases searching	See, for instance, Bioccelerator™ by Compugen, that performs similar operations [11]
7. High energy physics	Pattern searches, track finding, neural networks simulating	

Table 1 is not complete and other applications, not mentioned here, are also possible.

The board is intended for implementing specialized data processing functions and in this way its performance compares to that of some application specific boards. For example, Bioccelerator™ is an FPGA-based board family designed by Compugen for running rigorous searches of protein and nucleic acid sequence databases; the optimized logical circuitry runs search algorithms at speeds that are up to 1000 times faster than today's high-end computer workstations [11]. The performance of the PFPC-1V card, using Altera FLEX 10K130 devices for BPU implementation, compares to the performance of Bioccelerator-4. Another example is FT-2106x SHARC-based board family designed by Alacron; the FT-2106x cards are members of Alacron's family of high performance computing subsystems based on Analog Devices' ADSP-2106x SHARC (Super Harvard ARchitecture Computer) processor. Any FT-2106x card incorporates an array of up to eight SHARC processors and delivers high performance for DSP, image processing, pattern recognition and other applications [12]. PFPC-1V performs some operations, 2D convolution for instance, at speed comparative to the speed of FT-2106x with eight SHARC processors. The comparison is made exclusively for fixed point operations, while FT-2106x can also handle floating point numbers.

For comparison, Table 2 shows the times required to perform some specific image processing algorithms on the Parallel FPGA Processor Card and for representative PC and workstation platforms.

**Table 2.**

<b>Algorithm</b>	<b>Pentium- 100 Mhz 16 Mbyte RAM</b>	<b>PentiumII- 333 Mhz 128 Mbyte RAM</b>	<b>Hyper SPARC 64 Mbyte RAM</b>	<b>PFPC-1V 33 Mhz 4 Mbyte RAM</b>
Convolution With filter 4x4	0.65s	0.11s	0.76s	0.02s
Median Filter 3x3	1.97s	0.49s	0.75s	0.001s
Sharpening	0.51s	0.13s	1.31s	0.004s
Comparing with mask 32x32	43.78s	7.14s	58.89s	0.142s
Matrix multiplication	8.61s	0.60s	12.31s	0.011s

Note for Table 2: Image size is 256x256 bytes;

## **Acknowledgments.**

We thank Joe Holland for the careful reading and remarks improving the article.

## References.

### Printed:

1. Gokhale et al., «Building and Using a Highly Parallel Programmable Logic Array», *Computer*, No.1, Jan. 1991, pp. 81-89.
2. Regler (ed.), «Data Analysis Techniques for High-Energy Physics Experiments», Cambridge: Cambridge University Press, 1990.
3. Aubury et al., «Advanced Silicon Prototyping in a Reconfigurable Environment», IOS Press, 1998.
4. Denby, «Neural Networks and Cellular Automata in Experimental High Energy Physics», *Computer Physics Communications*, Vol. 49, 1988, pp. 429-448.
5. AMPP Catalog, June 1998, by Altera Corporation.
6. S. Sidorov , «Data in DST – Prefix Access in Postfix Language», *Proceedings of EuroForth 97*, Oxford.
7. Frantov, M. Shumakov «DED – DST Editor and Debugger», *Proceedings of EuroForth 97*, Oxford.
8. R. Goslin, «A Guide to Using Field Programmable Gate Arrays (FPGA) for Application-Specific Digital Processing Performance», Xilinx Inc., V.1.0, 1995.
9. Elder and S. W. Zucker. «Local Scale Control for Edge Detection and Blur Estimation», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 20, No.7, July 1998.

### Internet:

10. <http://www.research.digital.com/SRC/pamette/overview>
11. <http://compugen.co.il/products/bioc-t.html>
12. <http://www.alacron.com/hard/ft2106v01.htm>