Computing and Informatics, Vol. 28, 2009, 127–137

HIGHLY EFFICIENT TWIN MODULE STRUCTURE OF 64-BIT EXPONENTIAL FUNCTION IMPLEMENTED ON SGI RASC PLATFORM

Maciej Wielgosz, Ernest Jamro, Kazimierz Wiatr

ACK Cyfronet AGH
ul. Nawojki 11, 30-950 Krakow
&
Akademia Gorniczo-Hutnicza
Al. Mickiewicza 30, 30-059 Krakow
e-mail: {wielgosz, jamro, wiatr}@agh.edu.pl

Manuscript received 29 June 2008

Abstract. This paper presents an implementation of the double precision exponential function. A novel table-based architecture, together with short Taylor expansion, provides a low latency (30 clock cycles) which is comparable to 32 bit implementations. A low area consumption of a single exp() module (roughtly 4 % of XC4LX200) allows that several modules can be implemented in a single FPGAs.The employment of massive parallelism results in high performance of the module. Nevertheless, because of the external memory interface limitation, only a twin module structure is presented in this paper. This implementation aims primarily to meet quantum chemistry huge and strict requirements for precision and speed. Each module is capable of processing at speed of 200 MHz with max. error of 1 ulp, RMSE equals 0.62.

Keywords: HPRC (High Performance Reconfigurable Computing), FPGA, elementary function, exponent function, RASC (Reconfigurable Application-specific Computing)

1 INTRODUCTION

Rapid growth of HPRC (High Performance Reconfigurable Computing) platforms (e.g. SRC-computers, DRC-computers, Cray, XtremeData) has encouraged scien-

tists and companies to develop libraries of hardware modules. It is worth noticing that the idea behind specialized hardware modules commonly known as IP-cores is quite similar to the modern vision of portable hardware modules intended to support super computer systems. However, hardware modules employed in HPRC systems are subjected to rougher limitations of the silicon area. This factor strongly impacts the number of modules that can fit into FPGA which in turn affects the overall data transfer between software and hardware parts of the system. HPRC modules should cope easily with the rest of the software as well as hardware parts of the system, therefore some unified interface is indispensable. HPRC have other important advantages over HPC (High Performance Computing) like significantly lower power consumption and more efficient silicon coverage (80 % of silicon area is utilized whereas processors usually do not exceed more then 11 %). Unfortunately, carrying out a floating-point operation within FPGA absorbs much more area than fixed-point calculations, therefore for a long time, FPGAs had not been employed to support double precision operations. Nowadays, there are some implementations of single precision floating-point operations [1, 2, 3]. Since the proposed exp() module aims at speeding up HPC chemistry and physics calculations, it has to be compatible with the data format employed so far. Consequently, the IEEE-754 double precision standard is adopted.

2 ALGORITHM CONSIDERATIONS

Floating point elementary functions evaluation methods can be separated into two groups: iterative [4] and non-iterative algorithms. Since iterative methods have longer computation latency, non-iterative methods are considered hereon. Non-iterative methods can be classified as follows:

- direct Look-Up Table (LUT)
- polynomial approximations
- a mixture of LUT and polynomial approximation.

Direct LUT methods are conceptually the simplest and their implementations result in very high performance. These methods also provide precise results. The most significant disadvantage of direct LUT methods is their huge memory size which rises exponentially with the width of input data. Consequently the area occupied by the memory becomes unacceptable when the input data width reaches roughly 16 bits.

An alternative solution to the LUT-based method is the polynomial approximation which is commonly employed in hardware. It has the several important advantages over the direct LUT-based algorithm. First of all, the implementation of this method consumes scientifically less resources and often leads to comparable speed achievements. The designer must however pick up a polynomial function that perfectly suits a task, otherwise approximation errors are unacceptable. The above considerations apply to single precision calculations, usage of these methods

with higher precision data imposes more strict restrictions to the approximating function and results in a much higher degree of the polynomial. The degree of the polynomial corresponds directly to the number of multiplications. Moreover, increased data precision implies also a wider data word which results in much more area occupied by multipliers. Summing up, for higher precision data (e.g. 64-bit) polynomial approximation methods lose their primary advantages.

An alternative approach to the polynomial approximation is to divide the input space into sections. Each section can be separately approximated with a lower degree polynomial. An increase on the number of sections results in a decrease of the polynomial degree. Conversely, every section has different polynomial coefficients that have to be stored, which in turn results in an increase of the memory area [5]. Therefore, the number of sections is a trade-off between the polynomial degree (the area of arithmetic units) and the memory size.

It has been decided to employ an algorithm that combines the positive features of both above mentioned techniques – LUT-based method and polynomial approximations. Many mixed method implementations have been introduced [1]. There were, however, mostly single precision solutions. First, there appeared a software implementation of mixed methods [6], then an early hardware application derived much from them and can be regarded as an attempt to adopt software algorithms to hardware. This approach does not take full advantage of hardware capabilities, therefore new hardware LUT-based algorithms of exp() evaluation [3] were proposed.

3 ARCHITECTURE OF EXP MODULE

The main idea behind the exp() module is based on the following equations [9, 10]:

$$e^x = 2^x * \log_2(e) = 2^{x_i} * e^{x - x_i / \log_2 e} \tag{1}$$

and

$$e^{x+y} = e^x * e^y \tag{2}$$

where x_i is an integer part of $x * \log_2 e$.

A mixed method adoption always leads to the dilemma of the trade-off between the LUT memories' size and the polynomial part of the algorithm. In case of exp() function implementation, increase of LUT size results in a decrease of the multiplication at the expense of the rising number of occupied LUT blocks. Nevertheless, the analysis of the resources occupied by multipliers and LUT memories has led to the conclusion that the employment of Block RAMs (BRAMs) embedded in the FPGAs would be a best solution. The replacement of floating-point multipliers with fixed-point ones further reduces occupied FPGA resources. It is possible to do so because the input data was previously converted into a fixed-point format. Furthermore, input data smaller than $2^{(-60)}$ may be neglected during the calculation as they have unnoticeable impact on the final result.

It should be noted how negative numbers are handled. The negative input argument $-x$ is usually split into integer part $-x_i$ and fractional part $-x_f$. Therefore,

the fractional part LUTs must also service the negative numbers, which results in one extra address bit of every LUT reserved for the sign. A better solution is obtained when negative numbers are limited only to the integer part (most significant bits part) [7]. Therefore for $x_f \neq 0$ and $x < 0$ the following mathematical identity is employed:

$$-x = -x_i - x_f = -(x_i + 1) + (1 - x_f) \tag{3}$$

It should be noted that the above identity is obtained by converting an input argument from the sign-and-magnitude format (floating point mantissa standard) into the two's complement format. In order to further reduce hardware requirements, the polynomial approximation is employed. According to Taylor-Maclaurin expansion:

$$e^x = 1 + x + x^2/2 + x^3/6 + \ldots \tag{4}$$

In order to disregard $x^2/2$ and higher degree expressions, the input argument must be very small to satisfy the maximum mantissa error $< 2^{-54}$ for the double precision format. This is satisfied for $x < 2^{-27}$. Consequently, the most significant 27 bits of input x_f (fractional part of x) are calculated employing LUT-based methods, the reset less significant bits are calculated using Taylor-Maclaurin expansion limited to: $e^x \approx 1 + x$. To obtain the final result, the results of the LUT-based algorithm and polynomial approximation are multiplied according to Equation (2). The employment of polynomial approximation results in a significant decrease in the number of multipliers and LUTs.

Summing up, the input argument x after the conversion to the fixed-point format is divided into 5 sections:

1. integer part (11-bit), x_I, which evaluates 2^{X_i} (exponent part of the result),
2. fractional MSB part, x_M, bits $2^{-1} \div 2^{-9}$,
3. fractional middle-bits part, x_D, bits $2^{-10} \div 2^{-18}$,
4. fractional LSB part, x_L, bits $2^{-19} \div 2^{-27}$,
5. fractional Taylor part, x_T, bits $2^{-28}\ldots$

Summing up, the following mathematical operations are employed:

$$x_I = \lfloor x * \log_2(e) \rfloor \tag{5}$$

$$x_F = x_M \& x_D \& x_L \& x_T = x - x_I \cdot (\log_2(2))^{-1} \tag{6}$$

$$y = 2^{x_I} * e^{x_M} * e^{x_D} * e^{x_L} * (1 + x_T) \tag{7}$$

where: & – bit concatenation, $\lfloor x \rfloor$ – rounding to the greatest integer x_I such that $x_I \leq x$. Using Equations (5) and (6) enables the separation of the integer and fractional parts. This step can be considered to be a scaling process that transforms input data to an interval of boundaries at 0 and ln(2). As x_I is a small integer, this approach in practice replaces a large multiplication (required by identity $e^x =$

$2^{X/\ln(2)}$ by two smaller multiplications, one to compute x_I, the second to compute $x_I \cdot \ln(2)$.

It is worth noticing that the above modules work in parallel. This approach required an additional couple of pipeline registers to adjust the different latency of these modules.

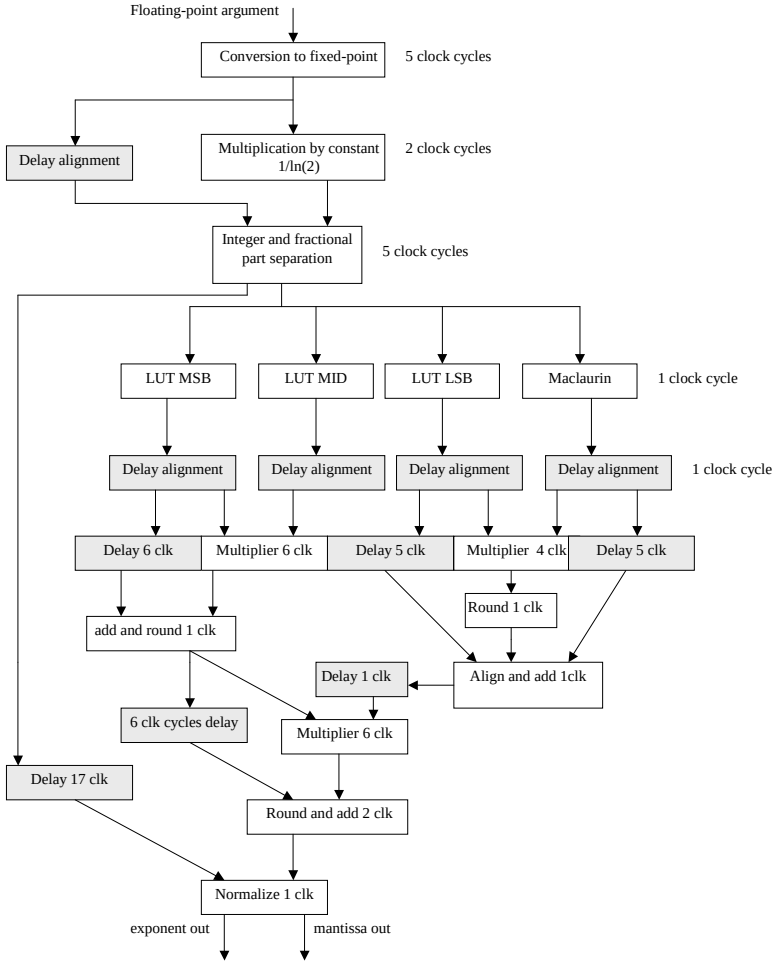


Fig. 1. Exp() module block diagram

3.1 Reduced Arithmetic Width

The multiplier inputs are roughly 60-bit wide, therefore the product width is 120-bit wide. Such a bit-width is far beyond the required precision, therefore the LSBs of

the product are usually disregarded. Consequently, an LSBs part of the multiplier performs operations which are not used in the next steps. As a result, in the proposed architecture, some of the LSBs logic is not implemented at all. Figure 2 illustrates the proposed approach for a 4×4-bit multiplier for which arithmetic for 2 LSBs (p_0, p_1) is not implemented. This approach allows for significant area reduction as it will be proved in the implementation results. Unfortunately, the calculation error is greater for the given architecture. The proposed architecture does not use any advance error compensation logic. The maximum error is equal the number of broken carry-in (ck-1 in 2) paths. To decrease this error, some additional guard bits are usually provided, e.g. in Figure 2 for the 4-bit output width (bits $p_6 \div p_3$) an additional logic for bit p_2 is implemented. It should be noted that for the FPGA implementation a modified multiplier architecture is used according to [8].

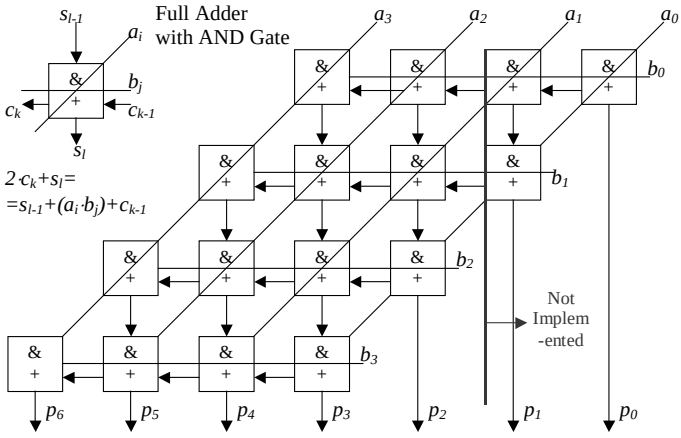


Fig. 2. Parallel-array multiplier with arithmetic width reduction

4 ERROR ANALYSIS

In the proposed architecture the following sources of errors can be distinguished:

1. Taylor series expansion,
2. multiplier (and LUT) width limitation:

   (a) The Taylor series expansion is limited only to: $e^x \approx 1 + x$. For $x \Longrightarrow 0$ the expansion error can be approximated by the next omitted expression, i.e. $x^2/2$. As input argument $x_T < 2^{-27}$ the Taylor series expansion error is limited roughly by 2^{-55}. It should be noted that the input value is $x_T \geq 0$, thus the result $y_T = 1 + x_T \geq 1$. Consequently, the relative error is also $\leq 2^{-55}$. Summing up, the Taylor series expansion error is much lower than the double precision format accuracy.

(b) In order to reduce hardware requirements for multipliers, parts of the LSBs are not implemented. This results in an additional calculation error. This error is limited by the number of carry paths which are cut. To limit this error, additional guard bits are introduced.

5 IMPLEMENTATION RESULTS

The presented implementation results contain an exp() module logic consumption together with RASC core services, essential to provide a compatibility with Altix 4700.

Implementation results	# 4-input LUT	# flip-flops	# 18-Kb BRAMs
Single exp() module	13.614 (7 %)	19.704 (11 %)	29 (8 %)
Twin exp() module	17.897 (10 %)	25.461 (14 %)	35 (10 %)

Table 1. Implementation results

Max. frequency	200 Mhz
Max. error	1 ulp
Root mean square error	0.61
Pipeline latency	30 clk cycles

Table 2. System parameters

6 APPLICATION PLATFORM

There are several reasons for choosing the SGI RASC platform as an implementation environment. First of all, RASC's platform delivers huge computational power. Secondly, SGI provides handy programming tools which, however, does not prevent the user from accessing low level code layers to carry out modifications (allow VHDL coding style) and are convenient for designers familiar with hardware design. The architecture of the SGI Altix 4700 belongs to the modern global shared memory group of super computing systems together with Cray XT4 and SRC-7.

SGI RASC RC100 Blade consists of two Virtex-4 LX 200 FPGAs, with 40 MB for each FPGA of SRAM logically organized as two 16 MB blocks (as shown in Figure 1) and an 8 MB block. The SRAM are 36-bit QDR devices with 4-bit parity, thus transferring 128-bit data every clock cycle. The RC100 Blade is connected using the low latency NUMALink interconnect to the SGI Altix 4700 Host System, for a rated peak bandwidth of 6.4GB per second. 128-bit data vectors are read from the MEM0, spread into two substreams each consisting of 64-bits. Every clock cycle (due to pipelining) data is processed by two exponential modules and results are concatenated to a 128-bit vector which finally is written to the MEM1. Afterwards the result is transferred through the NUMAlink to the rest of the system.

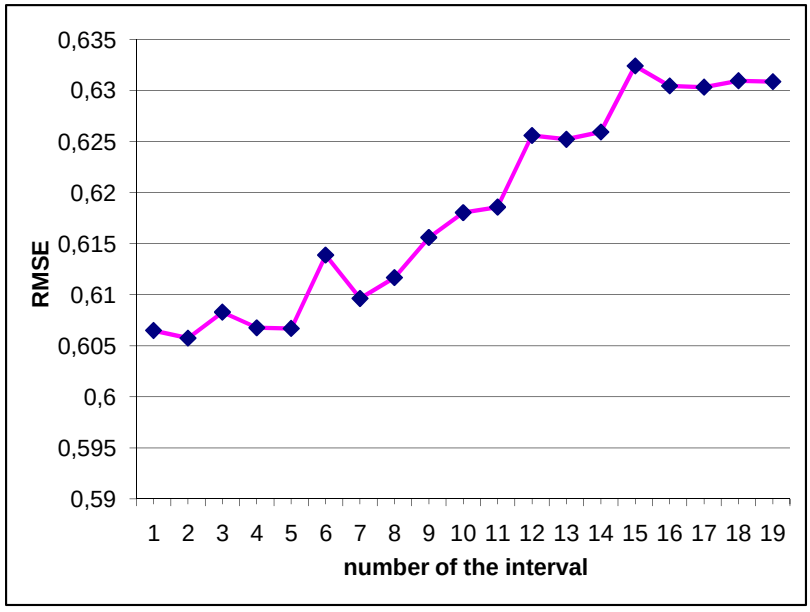


Fig. 3. RMSE across different ranges of input data

The RASC platform provides two FPGA chips (Xilinx Virtex 4 LX200), which allows to double the calculation rate by employing a second FPGA (this is not taken into account in Figure 5).

To compare the calculation speed-up achieved by the RASC, the average double precision calculation time per single exp function is given in Table 3 for Pentium 4

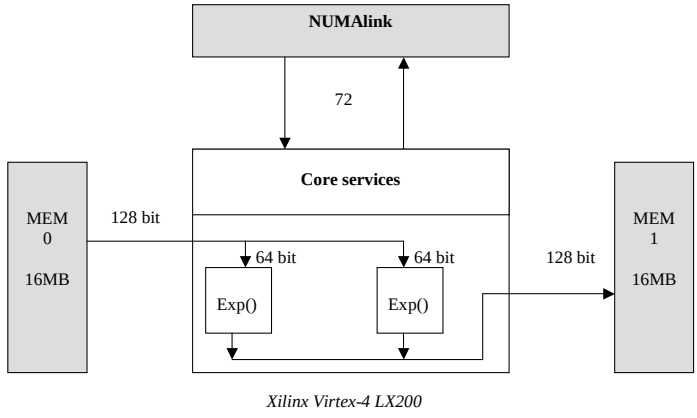


Fig. 4. System Overview of an FPGA on the SGI RC100 Blade

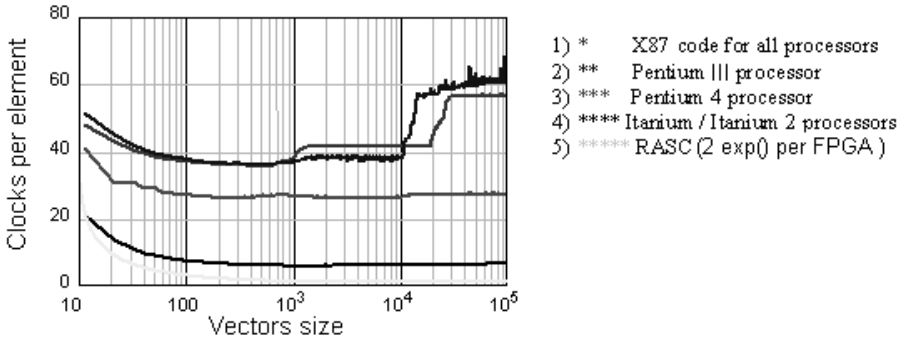


Fig. 5. Exp calculation on different platforms

	Pentium 4 processor	Itanium / Itanium 2 processors	RASC
Exp()	13.65	3.08	2.5

Table 3. Average calculation time [ns] per an exp calculation

and Itanium processors [8]. It is assumed that processors (Table 3) work at 2 GHz while single FPGA was clocked at 200 MHz.

The calculation speed-up achieved by the RASC is not significant, nevertheless it should be noted that the throughput can be doubled by employing two FPGAs – two FPGAs are incorporated in a RASC board. Secondly, only 10 % of FPGA resources are employed, thus additional arithmetic functions can be incorporated in the same FPGA. Besides, by improving external memory interface, the number of parallel exp() modules can be increased.

7 SUMMARY

This paper describes a novel architecture of double precision exponential function implemented in FPGAs and SGI RASC platform. The presented exp() architecture introduces several novel hardware solutions never used for the exp() function: a) 3 independent LUTs and Taylor series expansion for the exp() function, b) sign-migration to integer part, c) optimized multipliers.

There are two considerations on improvements worth introducing. The source code of quantum – chemistry software application can be thoroughly revisited in the future in order to eliminate a precision overhead. There is still a lot of silicon space on the FPGA (approximately 80 %) that can easily fit an additional logic. Investigations are being carried out to expand the exp() function with an additional logic of the hot spots found in the quantum chemistry application source code.

REFERENCES

[1] Doss, C. C.—Riley, R. L., Jr.: FPGA-Based Implementation of a Robust IEEE-754 Exponential Unit. 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '04), 2004, p. 229–238.

[2] Bui, H. T.—Tahar, S.: Design and Synthesis of an IEEE-754 Exponential Function. 1999 IEEE Canadian Conference on Electrical and Computer Engineering Shaw Conference Center, Vol. 1,1999, p. 450–455.

[3] Detrey J.—de Dinechin, F.: A Parameterized Foating-Point Exponential Function for FPGAs. IEEE International Conference on Field-Programmable Technology (FPT '05), Singapore, 2005, p. 27–34.

[4] Omondi, A. R.: Computer Arithmetic Systems. Prentice Hall. Cambridge, 1994.

[5] Lee, D.—Gaffar, A.—Mencer, O.—Luk, W.: Optimizing Hardware Function Evaluation. IEEE Transactions on Computers, Vol. 54, 2005, p. 1520–1531.

[6] Tang, P.: Table-Driven Implementation of the Exponential Function in IEEE Floating-Point Arithmetic. Argonne National Laboratory, ACM Transactions on Mathematical Software (TOMS), Vol. 15, 1989, p. 144–157.

[7] Wiatr K.—Jamro E.: Constant Coefficient Multiplication in FPGA Structures. Proc. of the IEEE Int. Conf. Euromicro, Maastricht, The Netherlands, Vol. 1, 2000, p. 252–259.

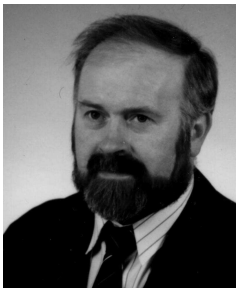
[8] Elzinga, S.—Lin, J.—Singhal, V.: Design Tips for HDL Implementation of Arithmetic Functions. Proc. of the IEEE Int. Conf. Euromicro, Maastricht, The Netherlands, 2000.

[9] Jamro, E.—Wielgosz, M.—Wiatr, K.: FPGA Implementation of 64-Bit Exponential Function for HPC. FPL Netherlands, August 27–29, 2007, FPL Proceedings.

[10] Wielgosz, M.—Jamro, E.—Wiatr, K.: Highly Efficient Structure of 64-Bit Exponential Function implemented. In FPGAs, ARC 2008 (Applied Reconfigurable Computing), March 26–28, 2008, London, UK.

Maciej Wielgosz Maciej Wielgosz obtained the M. Sc. degree in Electrical Engineering from the AGH University of Science and Technology in 2005. He is a Ph. d. student and the member of Reconfigurable Systems team. His primary research interests are in HPRC systems, image compression and neural networks. Currently involved in the project aiming at hardware implementation of quantum chemistry algorithms leading to deployment of the complete FPGA accelerator.

Ernest Jamro recieved M. Sc. degree in electronics engineering from the AGH University of Science and Technology (UST), Cracow Poland in 1996; M. Phil. degree from the University of Huddersfield (U. K.) in 1997; Ph. D. degree from the UST in 2001. He is currently a professor assistant in the Department of Electronics UST. His research interest include reconfigurable hardware (esp. Field Programmable Gate Arays – FPGAs), reconfigurable computing systems, System on Chip design, artificial intelligence.



Kazimierz Wiatr received the M. Sc. and Ph. D. degrees in electrical engineering from the AGH University of Science and Technology, Kraków, Poland, in 1980 and 1987, respectively, and the D. Hab. degree in electronics from the University of Technology of Łódź in 1999. He received the professor degree in 2002. His research interests include design and performance of dedicated hardware structures and reconfigurable processors employing FPGAs for acceleration computing. He received 9 research grants from Polish Committee of Science Research. These works resulted in 140 publications, including 3 books, the recent one: Acceleration Computing in Vision Systems. He is also an author of 5 patents and 35 industrial implementations. He currently is a Director of Academic Computing Centre CYFRONET AGH, and is a head of PIONIER council – Polish Optical Internet. And last but not least, he is a member of the Polish parliament (Senate), and a head of the Senate Science and Education Committee. `www.kazimierzwiatr.pl`.