Arithmetic Unit for Complex Number Processing

Dr. Solomon Khmelnik, Dr. Sergey Selyutin, Alexandr Viduetsky,
Inna Doubson, Semion Khmelnik

Abstract

This paper presents development of a complex number arithmetic unit (CAU), based on the single-component representation of complex numbers by positional binary codes with complex radix. Algorithms of the basic arithmetic operations in this representation are described and analyzed. The results of CAU design, simulation and synthesis and comparison of CAU characteristics to those of traditional arithmetic units (TAU) are presented. It is shown that implementation of CAU algorithms in math processors achieves significant (5-10 times) speed-up of complex number processing over TAU equivalents.

1. Introduction

Mathematical operations with complex numbers are commonly required in numerous computer applications, such as digital signal processing, wireless communication and telecommunication systems, control of power systems, etc. Algorithms for computer games and 3D graphics applications can be expressed using complex numbers. These operations usually include arithmetic operations, modulus comparisons, square and cube roots, logarithmic, trigonometric and hyperbolic functions, as well as fast Fourier transforms, to name a few.

Following historical mathematical notation, complex numbers in digital computers are represented as pairs of real numbers. Mathematic operations on complex numbers in TAU are implemented using operations on real numbers.

Consider operation of addition (subtraction) of two complex numbers:

$$(a + jb) \pm (c + jd) = (a \pm c) + j(b \pm d)$$

where j is an imaginary unit. This operation involves 2 real additions (subtractions).
Multiplication of two complex numbers:

$$(a + jb) \cdot (c + jd) = (ac - bd) + j(ad + bc)$$

will involve 6 real operations (four multiplications, one addition and one subtraction).
Division of two complex numbers:

$$(a + jb)/(c + jd) = (ac + bd)/(c^2 + d^2) + j(bc - ad)/(c^2 + d^2)$$

will involve 11 real operations (2 divisions, 6 multiplications, 2 additions and one subtraction). Similarly, it can be shown that operation of complex square root can be reduced to 6 real operations.

Due to representation of complex numbers with *separate* real and imaginary components, processing of complex numbers in TAU requires more systems resources and has much slower performance than real number processing.

To improve performance of complex number processing in computer systems several proposals have been made for a *single-component* representation. The essence of these proposals is to present complex numbers in such a positional number system, which allows for complex operations to be implemented on a single data element without a need for separate processing of real and imaginary components.

S. Khmelnik [2,5,9] was among the first to analyze and propose various positional codes for presentation and processing of complex numbers. He proposed and analyzed several positional coding systems, including those with $j\sqrt{2}$ radix and (-1+j) radix. Other early researchers in this field include D. Knuth [1] who suggested to use an imaginary radix $j\sqrt{2}$ for positional complex codes and W. Penney [4] who also suggested to use a complex radix (-1+j). In his works [2,5,6,7,9,14,15] S. Khmelnik suggested techniques for coding and decoding complex numbers and apparatus for arithmetic and mathematic processing of complex numbers, and then in [3,8,14-17] – implementation of these operations in digital hardware. Last works by S. Khmelnik [14-17] are being implemented in specialized ALUs for complex number processing.
Later several authors [11-13] suggested techniques for design of complex number multipliers. They used redundant codes for complex number representation, in order to achieve more regularity in ASIC hardware. However these coding systems were never applied to implementation of other arithmetic operations, such as division.

Six different complex radixes suggested in [2,5,9] by S. Khmelnik are discussed below in more detail.

2. Positional codes of complex numbers

In positional code systems the value of a real and complex number is expressed as weighted sum of radix powers multiplied by real or complex digits. The radix itself may be a positive, negative or complex (imaginary) number. The positional code of a complex number may be constructed by composition of positional codes of its real and imaginary components, presented in negative radix [9].

Consider a complex number Z = $X_\alpha + jY_\beta$. Let X_α and Y_β be its real and imaginary components presented in a radix ρ = -2 number system as:

$$X_\alpha = \sum_{(m)} \alpha_m \rho^m \quad \text{and} \quad Y_\beta = \sum_{(m)} \beta_m \rho^m .$$

The following binary codes correspond to these presentations:

$$K(X_\alpha) = \{\alpha_m\} \quad \text{and} \quad K(Y_\beta) = \{\beta_m\}.$$

Consider two methods of composing these codes into a single code of complex number.

In the first method a pair of digits (α_m, β_m) is designated by a numeral λ_m, which makes radix (-2) code with digits $\lambda_m \in \{0, 1, j, 1+j\}$:

$$K(Z) = \{\lambda_m\}$$

As shown in [4], this code may be also viewed as a positional code of form:

$$Z = \sum_m \gamma_m \cdot f(\rho,\ m)$$

with binary digits:

$$\gamma_m = \begin{Bmatrix} \alpha_m, & \text{if m is even} \\ \beta_m, & \text{if m is odd} \end{Bmatrix}$$

and variable radix: $f(\rho,m) = \begin{Bmatrix} \rho^{m/2}, & \text{if } m \text{ is even} \\ j\rho^{(m-1)/2}, & \text{if } m \text{ is odd} \end{Bmatrix}$

In the second method digits $\{\alpha_m\}$ and $\{\beta_m\}$ are interleaved: $\{\beta_{m+1}\alpha_{m+1}\beta_m\alpha_m\beta_{m-1}\alpha_{m-1}\}$. Let designate now $\alpha_m = \sigma_{2m}$, $\beta_m = \sigma_{2m-1}$ and rewrite this sequence in another form:

$$\{\sigma_{k+2}\sigma_{k+1}\sigma_k\sigma_{k-1}\sigma_{k-2}\sigma_{k-3}\},$$

where $k = 2m$.

This sequence forms a following code:

$$K(Z) = \{\sigma_m\}$$

It is shown in [9] that this code corresponds to radix $\rho = j\sqrt{2}$ code of a complex number $Z = X_\alpha + j\sqrt{2}\ Y_\beta$ with binary digits $\sigma_m \in \{0,1\}$:

$$Z = \sum_{(m)} \sigma_m \rho^{\ m}$$

Two types of positional codes described above support basic arithmetic operations on complex numbers: addition, subtraction, multiplication and division.

In [2,5,9,14,15] some other positional complex codes were analyzed. Several radices for building positional binary codes of complex numbers are presented in **Table 1**.

Table 1. Radices used to build positional codes of complex numbers.

№	Radix	Code of 2
1	$f(\rho,m)=\begin{Bmatrix}\rho^{m/2}, & \text{if m is even}\\ j\cdot\rho^{(m-1)/2}, & \text{if m is odd}\end{Bmatrix},\ \rho=-2$	10100
2.1	$f(\rho,\ m)=\rho^{m},\ \rho=j\sqrt{2}$	10100
2.2	$f(\rho,\ m)=\rho^{m},\ \rho=-j\sqrt{2}$	10100
3.1	$f(\rho,\ m)=\rho^{m},\ \rho=(-1+j)$	1100
3.2	$f(\rho,\ m)=\rho^{m},\ \rho=(-1-j)$	1100
4	$f(\rho,\ m)=\rho^{m},\ \rho=\frac{1}{2}(-1+j\sqrt{7})$	1010

By far, usage of radix $f(\rho,m)=\begin{Bmatrix}\rho^{m/2}, & \text{if m is even}\\ j\cdot\rho^{(m-1)/2}, & \text{if m is odd}\end{Bmatrix},\ \rho=-2$ gives much simpler and more effective hardware implementation of complex arithmetical unit.

The described coding system with complex radix have a number advantages compared to conventional coding systems with real radix:

- unified complex 2n-bit code with (-2) radix and interleaved imaginary and real bits vs pair of two n-bit codes in radix 2 for real and imaginary part
- the sign operations are not required due to (-2) radix
- there is no need for 2's complement transformations
- rules of overflow and underflow detection are simplified
- algorithms of operations with variable length codes are simpler
- "digit-by-digit" technique can be used for computation of all elementary functions of complex arguments

3. Basic operational blocks of complex arithmetic unit

3.1. Addition and Subtraction

Taking into account the described methods of positional coding, algebraic addition of complex numbers in these systems is performed according to the rules of algebraic addition in positional codes of real numbers with (-2) radix. **Fig. 1** illustrates the pattern of summation and carry propagation in the process of algebraic addition of single-component complex codes.

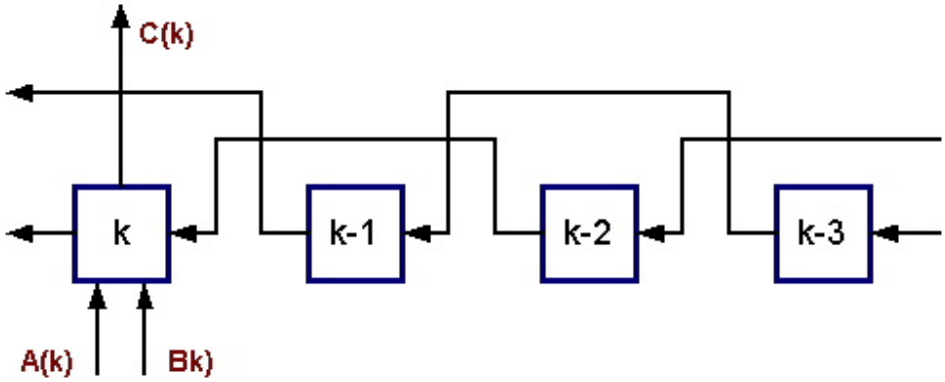

Fig. 1. Carry Propagation Pattern in Coding Systems 1 and 2

Although complex numbers are represented by single-component codes, due to described [9,15] properties of coding systems even and odd digits are processed separately and in parallel, which amounts to the same speed of operation and double length of operands compared to traditional adders for real numbers. It was shown [9] that carry propagation signals can be represented by two bits in all binary complex number systems with complex radices.

3.2. Multiplication and Division

Multiplication of two complex numbers:

$$V = \sum_k v_k f(\rho, k) \text{ and } W = \sum_h w_h f(\rho, h)$$

results in a product determined as:

$$Z = \sum_m \left[V w_m f(\rho, m) \right]$$

Multiplication by the basic function $f(\rho, h)$ is equivalent to the h-digit shift of the code with radix ρ. Since $w_h = \{0,1\}$, the multiplication of codes in this number system is reduced to the series of additions and shifts. **Fig. 2** shows a traditional sequential multiplication circuit, which is also applicable to the multiplication of complex numbers.

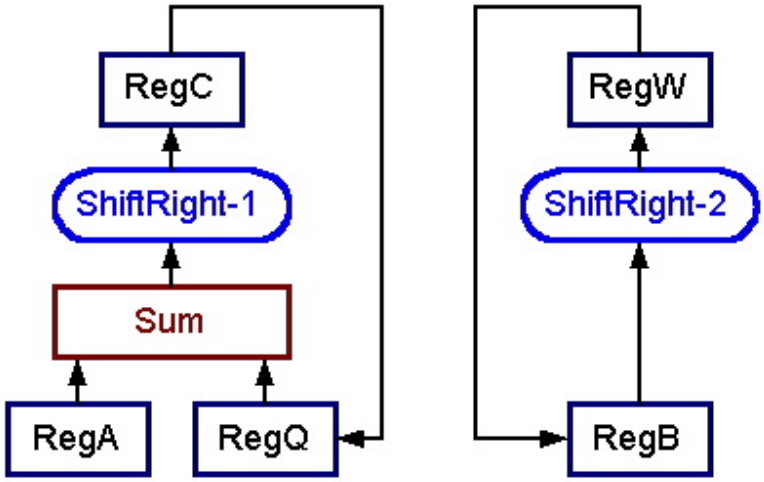

Fig. 2. Sequential Multiplication Complex Numbers

The major differences from traditional multiplication of real numbers are absence of sign operations and execution of operation on the codes as a *whole*, without *separate* processing of real and imaginary components. Multiplication performance and speed is determined by the number *m* of analyzed digits at each "add-and-shift" cycle. For the two types of coding systems, having m=2 requires an adder circuitry to support addition (subtraction) of X, $-X$ and $-2X$ multiples [15].

Division using proposed code systems is somewhat similar to traditional division of real numbers, with the following exception. The goal of division is to converge remainder to zero. In traditional algorithms the sign of remainder is analyzed and is used to determine value of next subtraction (addition). In our case analysis of complex modulus is performed to achieve the same goal. Once again, division operation is executed as a *whole*, without need for *separate* processing of real and imaginary components.

3.3. Elementary Functions of Complex Argument

Although it goes beyond the scope of this discussion. It is worth mentioning that "digit-by-digit" technique for computation of elementary functions of real arguments [10] may be extended on complex arguments as well. Algorithms for computation of complex elementary functions were developed in [9,15] and include trigonometric and hyperbolic functions, logarithm and antilogarithm, raising to a complex power, modulus calculation, finding square root, transformation between polar and Cartesian coordinates, vector rotation.

3.4. Complex Arithmetic Unit Architecture

In order to support described operations CAU will include such major blocks as adder, multiplier etc, auxiliary blocks to support execution of miscellaneous unary and binary operations and microprogram block to implement computation of elementary functions. The overall architecture of CAU is shown at **Fig. 3**.

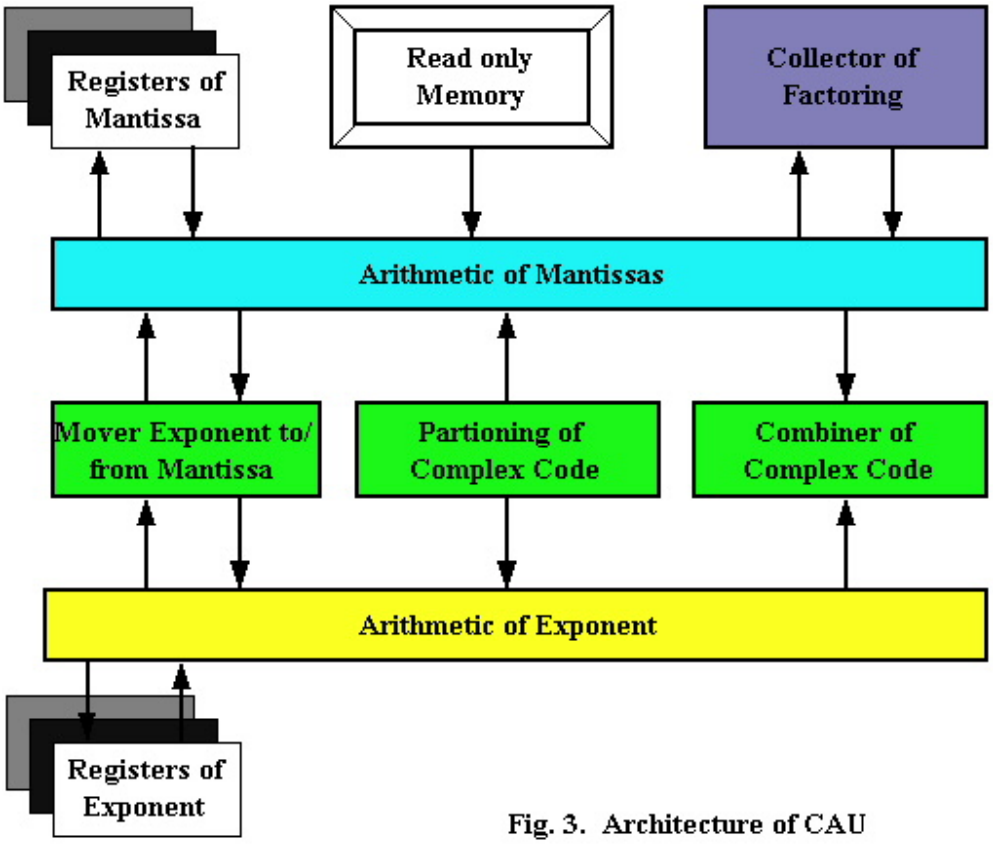

Fig. 3. Architecture of CAU

4. Comparison with traditional arithmetic unit

It was demonstrated earlier that complex operations in TAU are implemented using operations on real numbers. We will show that implementation of complex operations in CAU requires more or less the *same* amount of time consumed by their real equivalents in TAU.

Traditional AU will use four n-bit real codes to represent two complex numbers with n-bit precision and CAU will use two 2n-bit complex codes to represent two complex numbers with n-bit precision. Due to embedded parallelism in processing of even and odd bits, CAU will perform a complex addition of these codes at the speed of *one* n-bit real addition, whereas TAU will require two n-bit real additions to support the same operation of complex addition.

In the case of addition of floating-point numbers, a better precision vs. traditional approach is achieved due to common exponent for real and imaginary parts. It can be shown that for 64-bit codes this is equivalent to 14% savings in bit length vs. traditional approach, where separate exponents are used in representation of floating-point reals and imaginaries.

Similarly, it can be shown that complex multiplication of two 2n-bit complex codes in CAU can be performed at the speed of *one* multiplication of n-bit real codes. At the same time TAU will require *four* multiplications of n-bit real codes and *two* additions of n-bit real codes.

In general, a complex arithmetic operation on 2n-bit codes in CAU takes the same time as corresponding real arithmetic operation on n-bit codes in TAU. To speed up performance of math operations even further traditional hardware acceleration techniques such as carry-save/ripple-carry adders and matrix multipliers can be implemented in CAU, resulting in increased system frequency.

The following table shows how many real operations are required for software implementation of some complex algebraic operations in TAU.

Table 2. Comparison of TAU versus CAU performance.

Operation	NoIn			NoALU clocks			NoCPU clocks		
	tTAU	tCAU	TAU/ CAU	tTAU	tCAU	TAU/ CAU	tTAU	tCAU	TAU/ CAU
Algebraic addition of integer complex numbers	**2**	**1**	**2**	**2**	**1**	**2**	**20**	**10**	**2**
Multiplication of integer complex numbers	**6**	**1**	**6**	**6**	**1**	**6**	**60**	**10**	**6**
Affine integer transformation	**8**	**1**	**8**	**8**	**2**	**4**	**80**	**13**	**6**
Basic operation for integer FFT	**8**	**1**	**8**	**8**	**2**	**4**	**80**	**19**	**4**
Algebraic addition of floating complex numbers	**2**	**1**	**2**	**12**	**6**	**2**	**30**	**15**	**2**
Multiplication of floating complex numbers	**6**	**1**	**6**	**16**	**2**	**8**	**70**	**11**	**6**
Affine floating transformation	**8**	**1**	**8**	**16**	**2**	**8**	**88**	**13**	**6**
Basic operation for floating FFT	**8**	**1**	**8**	**16**	**2**	**8**	**88**	**19**	**4**
Division of integer	**11**	**1**	**11**	**204**	**80**	**3**	**303**	**89**	**4**

complex numbers									
Comparison of complex number modules	**5**	**1**	**5**	**85**	**1**	**85**	**130**	**10**	**13**
Functions of a complex argument	**~20**	**1**	**~20**	**~900**	**~80**	**11**	**~1100**	**~90**	**12**

***NOTE**. Based on CAU architecture it is possible to develop math processors for matrix computations and geometric transformations. It is estimated that such processors will speed up performance of FFT by at least 7 times and of affine transforms by at least 20 times.*

Table 2 given below shows comparison of CAU and TAU performance. It is based on the results of TAU and CAU simulation. In this table the following notations are used:

NoIn - number of instructions to implement operation. It is assumed that multiplications and affine transformations, as well as basic FFT operations are performed using array multipliers
NoALU - number of ALU clocks to execute operation
NoCPU - number of CPU clocks to execute all instructions for given operation, including memory access, which is equivalent to 3 ALU clocks
tTAU - parameters of traditional ALU for real numbers (TAU)
tCAU - parameters of ALU for complex numbers (CAU)
TAU/CAU - ratio of TAU speed over CAU speed

This comparison is based on number of operations instead of operation execution times, because those depend largely on system frequency and processor technology, rather than on processor architecture.

Assuming that algebraic operations are distributed uniformly across math applications, one will conclude that each complex operation in TAU requires in average 5 real operations. Considering also implementation of geometrical and fast Fourier transformations as complex operations, will result in 5-10 times speedup of complex number processing in CAU compared to TAU.

This technology significantly reduces the number of required computer operations at a small cost of increased circuit complexity. Such reduction of operations leads to a faster computation time and should cause power savings.

5. Conclusion

At the current stage of development the following results were achieved:

1. sound theoretical foundation of complex computer arithmetic and methematics, based on proposed unified code for complex number presentation
2. design and development of CAU hardware, implementing a vast class of math operations and functions on complex numbers
3. development of VHDL simulation models of CAU blocks

4. synthesis of CAU operational and control blocks, targeted for different ASIC and FPGA technologies
5. simulation and testing of CAU functional behavior and performance in the system environment, using behavioral and RTL models

In addition, the following tools and models were developed for CAU simulation, debugging and testing:

6. VHDL model of virtual processor and memory, presented at **Fig. 4**
7. programming language for this processor

- several application programs
- browsers for viewing the architecture and layout of design blocks, algorithms of CAU and functional tests

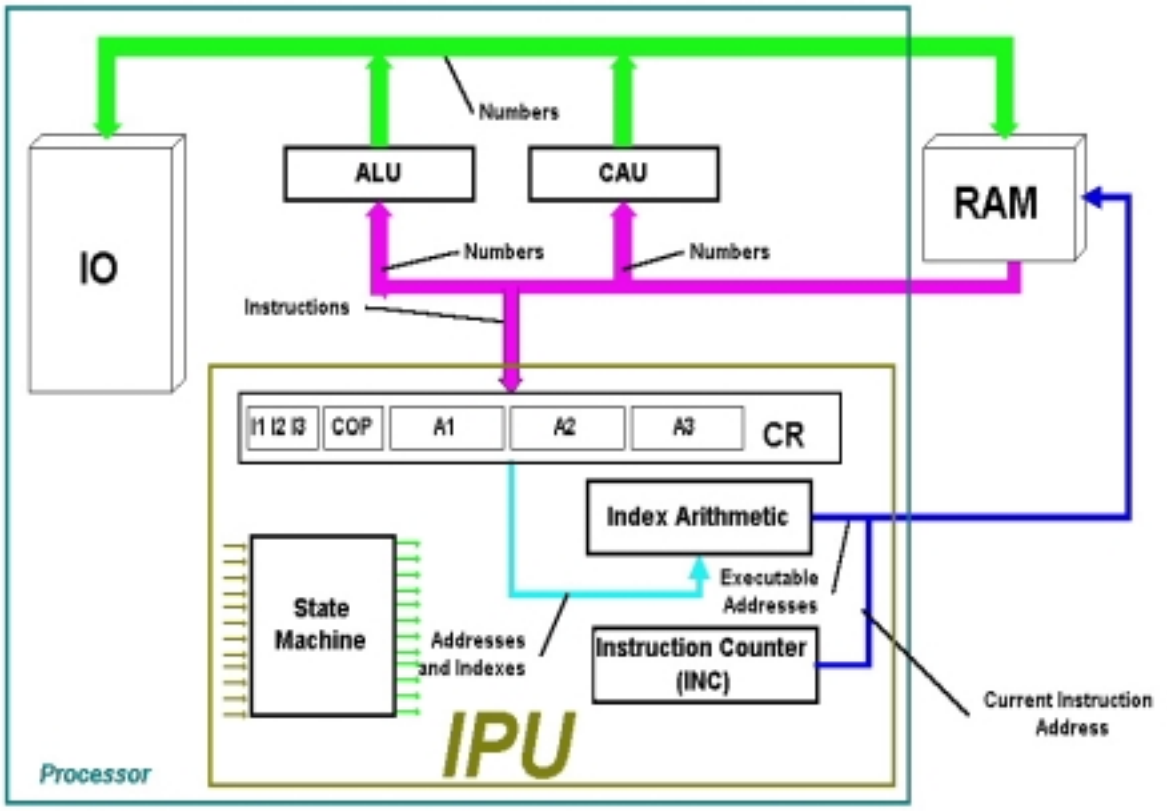

Fig. 4. Architecture of virtual CPU with CAU

Based on results of design, simulation and synthesis the following technical parameters of CAU were established:

- number of operational blocks ~ 190
- number of individual operations ~ 250
- number of logic gates ~ 250,000
- minimal system clock period ~ 30 gate delays

The future stages of CAU development will include further enhancements in math processor architecture and tools to support effective implementation of computer applications with intensive complex number processing.

Building math processors, based on described technology, will make possible to achieve the following results in applications with complex numbers:

- performance of complex numbers processing increased by ~ 5-10 times
- data memory requirement reduced by ~ 15%
- labor-intensiveness of programming reduced by ~ 2 times

6. References

1. D.E. Knuth. “An Imaginary Number System”, Communications of ACM 3, No. 4, 1960.
2. S. Khmelnik. “A Specialized Computer for Operations on Complex Numbers”. Questions of Radio Electronics XII, No. 2, 1962 (in Russian).
3. S. Khmelnik. “Adder of codes of complex numbers”. Questions of Radio Electronics XII, No. 3, 1965.
4. Penney W., “A ‘Binary’ System for Complex Numbers”, Journal of ACM 12, No. 2, 1965, pp. 247-248.
5. S. Khmelnik. “Positional Coding Systems for Complex Number Presentation”. Questions of Radio Electronics XII, No. 9, 1966 (in Russian).
6. S. Khmelnik. “Number system with complex bases”, Part in Book: Pospelov D.A., Arithmetic basis of digital Computers, "Visshaja shkola", 1970, Moscow.
7. S. Khmelnik. “Solution of the navigation problems on a digital computer using complex numbers codes”, "Problem special radio-electronic", series "Tele-mechanics and management Systems", 1971, No 6, Moscow.
8. S. Khmelnik et al. “Digital devices with microchips”, "Energia", 1975, Moscow.
9. S. Khmelnik. “Computer Arithmetic of Vectors, Figures and Functions”, Mathematics in Computers, 1995, Tel Aviv (in Russian).
10. Miller J. M., “Elementary Functions. Algorithms and Implementation”. Birkhauser, 1997, Boston.
11. T. Aoki, H. Amada, and T. Higuchi: "Real/Complex Reconfigurable Arithmetic Using Redundant Complex Number Systems". In Proc. 13th Symposium on Computer Arithmetic, 1997.
12. Y. Chang and K. Parhi. “High Performance Digit Serial Complex Number Multiplier-Accumulator”, Proc. Int. Conf. on Computer Design,1998.
13. A. M. Nielsen and P. Kornerup, “Redundant Radix Representation of Rings”, IEEE Transactions on Computers, Vol. 48 (11), November 1999
14. S. Khmelnik, “A Method and System for Processing Complex Numbers”. International Patent Application, WO 01/50332, 2001. . United States Patent Application No. 10/189,195, 2002.
15. S. Khmelnik, “A Method and System for Implementing Coprocessor”. Canadian Patent Application, CA 02293953, 2000.
16. S. Khmelnik. “Method and System for Processing Matrices of Complex Numbers and FFT”, Canadian Patent Application, CA 2339919, 2001.
17. S. Khmelnik, “Method and System for Processing Matrices of Complex Numbers and Complex Fast Fourier Transformation”. International Patent Application, PCT/CA02/00295, 2002