Computation Model of *p*-adic Arithmetic

Borislav P. Stoyanov[1], Borislav Y. Bedzhev[2] and Zhivko S. Zhekov[3]

***Abstract*—The classical computing with floating point data presentation accumulates small errors at every step. As a consequence some times the final result of a complex program is entirely false. With regard to this, our paper suggests a computation model of *p*-adic arithmetic using exact Hensel code presentation in Visual C++ environment. It could be applied successfully for developing of stream ciphers with *Feedback with Carry Shift Register* architecture and for synthesis of pseudorandom sequences with variety of statistical properties.**

***Keywords*—Programming, Cryptography, Communication system signaling.**

I. INTRODUCTION

The classical computing algorithms, based on floating point data presentation, accumulate small errors at every step. As a consequence some times the final result of a long and complex program is much distorted or even so entirely false. This situation motivated active researches directed to finding more powerful and accurate computing tools able to interpret with insight a given science or technical problem. The efforts in this area leaded to developing of new methods of computer programming and data expressing. One of them proposed recently, is the method of the so-named *p*-adic arithmetic [1], [4], [6], [7], [8]. It seems to be very effective in some applications such as orthogonal signal synthesis, cryptography, spread spectrum systems and so on. The positive features of this method will be clarified with following example. Let us consider the recursive filter which output u_i in time moment $i.\tau$ (τ denotes the clock period of the filter) is:

$$u_i = u_{i-1} + u_{i-2};\ \ u_0 = u_1 = 1. \tag{1}$$

The output can be determined using two approaches. The simpler one consists in recursive (step by step) computing of the consecutive outputs u_2, u_3, …, u_i. This way needs *i* steps and hence a lot of time for large i. The second possible approach is that of using the Eq. (2):

$$u_i = \frac{1}{\sqrt{5}}.\left[\frac{(1+\sqrt{5})^i}{2} - \frac{(1-\sqrt{5})^i}{2}\right] \tag{2}$$

The Eq. (2) gives the result directly (in $\lfloor \log_2 i \rfloor + 1$ steps), but here a problem is the impossibility to represent the irrational number $\sqrt{5}$ exactly. This obstacle can be avoided applying the method of *p*-adic arithmetic. For instance, the number $\sqrt{5}$ has the following 11-adic exact representation (7606, 0) [7]. Hence, the filter output u_i can be calculated using Eq. (2) and 11-adic arithmetic in $\lfloor \log_2 i \rfloor + 1$ steps. The final result will be obtained with great accuracy after transforming in usual arithmetic.

With regard to the all above cited, this paper aims to suggest a computation model of *p*-adic arithmetic using exact Hensel code presentation in Visual C++ environment.

The paper is organized as follows. First, the basics of *p*-adic arithmetic are recalled. After then, the computation model of *p*-adic arithmetic using exact Hensel code presentation in Visual C++ environment is described. Finally, the advantages and possible areas of application of our model are discussed.

II. BASICS OF *P*-ADIC ARITHMETIC

We will explain the bases of the *p*-adic arithmetic refer to [1], [4] , [6].

For any positive integer *m*, denote $\mathbf{Z}_m$ the ring of integers modulo *m* and by |.| - the canonical ring homomorphism from $\mathbf{Z}$ to $\mathbf{Z}_m$. Let $\mathbf{N}$ be the set of natural numbers. For a given prime *p*, a rational number $\alpha = a / b$ can be represented in a unique way as is Eq. (3):

$$\alpha = (c/d) * p^e, \tag{3}$$

where *c*, *d* and *e* are integers, *c*, *d* and *p* pairwise relatively prime, *d* and *p* positive. This kind of representation of rational numbers is called the normalized form.

The function in Eq. (4)

$$\|.\|_p : Q \to R \tag{4}$$

[1]Borislav P. Stoyanov is with the Faculty of Mathematics, Informatics and Economics, Shumen University, 1 Universitetska Str., 9700 Shoumen, Bulgaria, E-mail: bpstoyanov@abv.bg

[2]Borislav Y. Bedzhev is with the Faculty of Artillery, Air Defense and Communication Information System, National Military University, 1 Karel Shkorpil Str., 9713 Shoumen, Bulgaria, E-mail: bedzhev@mail.pv-ma.bg

[3]Zhivko S. Zhekov is with Space Research Institute, Bulgarian Science Academy, PIO Box 118, 9700 Shoumen, Bulgaria, E-mail: zhekov@yahoo.com

from the rational numbers Q to the real numbers R, defined as in Eq. (5)

$$\| \alpha \|_p = \begin{cases} p^{-e}, if\ \alpha \neq 0 \\ 0\ \ , if\ \alpha = 0 \end{cases} \quad (5)$$

is a norm on Q, called the p-adic norm [4]. Furthermore α can be uniquely expressed in the following form in Eq. (6):

$$\alpha = \sum_{i \geq e} a_i p^i, \quad (6)$$

where $a_i \in \mathbf{Z}_p$. The infinite sequence $(a_e a_{e+1} \ldots a_{-1} a_0 a_{+1} \ldots)$ is called *p-adic representation* of α. The p-adic expansion of a rational number is periodic and it can also assume the following form: $\alpha = (a_e a_{e+1} \ldots a_{-1 \ldots} a_{k-m-1}{}' a_{k-m} \ldots a_{k-1} a_k)$, where the m digits from a_{k-m} to a_k constitute the period.

For instance, above mentioned 11-adic representation (7606, 0) of the irrational number $\sqrt{5}$ means that $a_0 = 7,\ a_1 = 6,\ a_2 = 0,\ a_3 = 6,\ a_4 = a_5 = ... = 0$ in Eq. (6).

We use *truncated representation*, defined as follows.

Definition *1* **(Hensel Code).** Given a prime number p, a Hensel code of length r of any rational number $\alpha = (c / d)*p^e$ is a pair of Eq. (7):

$$\left| c * d^{-1} \right|_{p^r} = \sum_{i=0}^{r-1} a_i \cdot p^i \in \mathrm{Z}_{p^r} \cdot \quad (7)$$

Let $\mathbf{H}_{p,r}$ denote the set of all Hensel codes where subscripts mean the prime p and the approximation r respectively, and let $\mathbf{H}(p, r, \alpha)$ indicate the Hensel code representation of the rational number $\alpha = (a / b)*p^e$.

The forward mapping between rational numbers and Hensel codes can then be defined on the bases of the following theorem.

Theorem *1* **(Forward Mapping)**. Given a prime p, an integer r and a rational number $\alpha = (c / d)*p^e$, such that c, d and p pairwise relatively prime integers, the mantissa $mant_\alpha$ of the code related to the rational number α, is computed by the Extended Euclidean Algorithm (EEA) [10] applied to p^r and d as: $mant_\alpha \equiv c \,.\, y \pmod{p^r}$, where y is the second output of the EEA.

Definition *2* **(Farey Fraction Set).** Let

$$\mathrm{N}_{(p,r)} = \left\lfloor \sqrt{\frac{p^r - 1}{2}} \right\rfloor \quad (8)$$

The Farey fraction set $\mathbf{F}_{p,r}$ of order $\mathbf{N}_{(p,r)}$ is the subset of rational numbers a / b such that: a,b $\in \mathbf{N}$, $0 \leq a \leq \mathbf{N}_{(p,r)}$, $0 < b \leq \mathbf{N}_{(p,r)}$.

Arithmetic operations on Hensel codes are carried out, digit by digit, starting for the leftmost digit, as in the usual base-p arithmetic operations. An addition (or subtraction) can give a result in which some leftmost digits are equal to zero. In this case we make the addition (or subtraction) produced in a *pseudo-Hensel code*.

Definition *3* **(Pseudo-Hensel codes)**. A pseudo-Hensel code is a code such that $a_0 = \ldots = a_k = 0$, for some k with $0 < k \leq r$-1.

Theorem *2*. Given a prime p, an approximation r, given an arithmetic operator Φ in Q and the related arithmetic operator $\Phi' \in \mathbf{H}_{p,r}$, if $\mathbf{H}(p, r, \alpha_1)\ \Phi'\ \mathbf{H}(p, r, \alpha_2) = \alpha_3'$, then there exists only one $\alpha_3 \in \mathbf{F}_{p,r}$, such that $\alpha_3' = \alpha_3$.

Now let us consider the arithmetic operations in $\mathbf{H}_{p,r}$.

Addition. Given two Hensel codes $\mathbf{H}(p, r, \alpha) = (mant_\alpha , exp_\alpha)$ and $\mathbf{H}(p, r, \beta) = (mant_\beta , exp_\beta)$, first of all we must remove the smaller mantissa to the right side in order to obtain $exp_\alpha = exp_\beta$. After that perform the addition taking into account that all the operations are carried out from left to right.

Subtraction. The subtraction could be performed using two approaches.

First, we can compute the complement *mod* p^r of the minuend and then to carry out the addition.

Second, if the minuend is a pseudo-Hensel code, then the subtraction can be carried out in the usual way, without using the complement of the minuend (except in the case when the subtrahend is the Hensel code which represents zero). In this situation in order to carry out the subtraction we must get a p-adic unit from the right digit (instead of from the left digit, as usually happens in subtraction between two integer numbers).

Multiplication. When we perform multiplication we must operate by multiplying the respective mantissas of the codes, and then we must add their exponents. Also in this case the code result is truncated to r digits.

Division. In order to perform division we must operate by dividing the respective mantissas of the codes, and then we must subtract the respective exponents.

If the first digit of the divisor is zero, we cannot compute the modular inverse as stated in the classical algorithm. Nevertheless we can carry on with the computation, because the code approximation has not been decreased, but if we want to compute a division in which the dividend belongs to $\mathbf{H}_{p,r}$ and the divisor is a pseudo-Hensel code of order k, with $k < r$. We can appropriately manipulate these codes with algorithm "Lim92" [7], in order to apply the division algorithm. In this way we can avoid the loss of significant digits and we can manipulate the pseudo-Hensel codes in the same way as the Hensel codes.

III. COMPUTATION MODEL OF *P*-ADIC ARITHMETIC

We propose a computation model of the p-adic arithmetic in exact Hensel code presentation in Visual C++ environment. The model is based on two files *p_adic.h* and *p_adic.cpp*. Our class *p_adic* has the following structure:

```
class p_adic : CObject{
public:
    long a;                    //numerator
    long b;                    //denominator
    long sign;                 //sign of the rational number
    long c;                    //numerator after elimination
    long d;                   //denominator after elimination
    long base;                 //p-adic base
    long ord;                  //exponent in the elimination
    long prec;                 //precision
    long cd_pr;                //c*d^-1 mod p^r
    CArray<int, int>mantisa;     // mantisa
}
```

The class *p_adic* has two constructors.

The first is *p_adic(CString &m, long e, long s, long p, long r)*. It creates expansion from *CString*, where *e* is the exponent, *s* – the sign of the exponent, *p* means the *p*-adic base and *r* – the approximation. For example, if the initial parameters are *m* = "4333", *e* = 2, *s* = -1, *p* = 5 and *r* = 4 then after execution of this constructor the next elements receive the values: *base* = 5, *ord* = -2, *prec* = 4, *mantisa* = (4, 3, 3, 3).

The second constructor is *p_adic(long x, long y, long p, long s, long r)*. It creates expansion from rational number, where *x* is the numerator, *y* – denominator, *p* – *p*-adic base, *s* – sign of the rational number, *r* - the approximation. If we have *x* = 4, *y* = 3255, *s* = -1, *p* = 3, *r* = 10 then the created object will have *a* = 4, *b* = 3255, *sign* = -1, *c* = 4, *d* = 1085, base = 3, *ord* = -1, *prec* = 10, *cd_pr* = 53389, *mantisa* = (1, 0, 1, 0, 2, 0, 1, 0, 2, 2).

The obtaining of the class elements depends on few auxiliary member functions.

The member function *void elimination(long &x, long &y,long &e, long p)* creates from a given rational number *a / b* the rational number from Eq. (3) with the help of the member function *long gcd(long x, long y)*. It returns the greatest common divisor [10] from *x* and *y* and *long fract_ord(long *x, long *y, long p)* returns the exponent *e* from Eq. (3).

The member function *long integer_part(long c, long d, long s, long pr)* calculates the long number *cd_pr* by computing of $c*d^{-1} \bmod p^r$, with the help of the member function *long inverse(long a, long n)*, which returns $a^{-1} \bmod n$.

The member function *void expansion(long cd_pr)* creates the *p*-adic expansion from *cd_pr* and put it in *mantisa*.

We have realized in the class *p_adic* the following four basic operations:

Addition - void sum(p_adic y, p_adic z)*. For example, if we have 34/4 + 2/7 in $\mathbf{H}_{2,5}$, the result is (1 0 1 1 0, -1).

Subtraction - void sub(p_adic y, p_adic* z)*. For example, if we have 15/9 – 2/28 in $\mathbf{H}_{3,7}$, the result is (2 2 2 2 1 0 0, -1).

Multiplication - void mul(p_adic y, p_adic* z)*. For example, if we have 50/7 * 2/23 in $\mathbf{H}_{7,6}$, the result is (1 2 1 5 2 6, -1).

*Division – void div_norm(p_adic *y, p_adic *z)*. For example, if we have 2/3 / (-51/9) in $\mathbf{H}_{7,9}$, the result is (4 6 4 3 4 1 1 6 2, 0).

The class *p_adic* with the *p*-adic member functions can be included as a part in projects with *p*-adic calculations. To illustrate this we propose an application "Hensel exact calculation" - hcalc.exe for executing *p*-adic arithmetic in Hensel code.

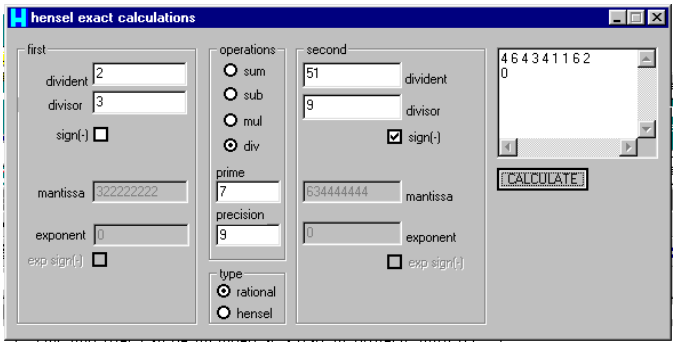

Fig. 1. Expansion from rational numbers

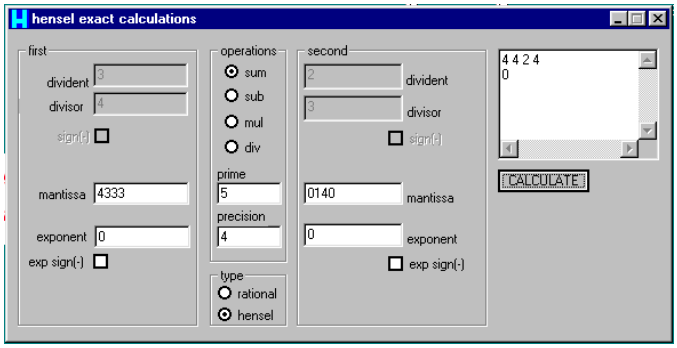

Fig. 2. Expansion from string

Here, according to the two constructors, we have two kind of *p*-adic expansion obtaining: first, from rational numbers as it is shown on Fig. 1 and second, from string (see Fig. 2).

IV. Conclusion

The advantages of proposed in our work *p*-adic computation modeling will be clarified by following example. Let us consider the modern stream ciphers, which are designed by combining the outputs of several *Linear Feedback Shift Registers* (*LFSR*). Here one new direction of development was proposed from Klapper and Goresky recently [5]. It will be explained using Fig. 3.

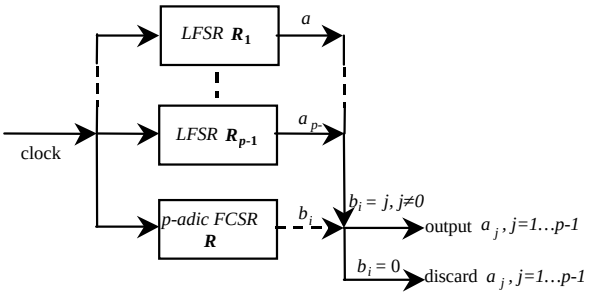

Fig 3. The Shrinking Generator with *p*-adic controlling *FCSR*

How it is shown, the Klapper & Goresky's stream cipher uses a controling *p*-adic *Feedback with Carry Shift Register (FCSR) R* to select a portion of the output sequences of *LFSRs* from R_1 to R_{p-1}. Therefore, the produced keystream is a *shrunken and mixed* version of the output sequences of *LFSRs* R_1 to R_{p-1} as it is specified in Fig. 3.

The algorithm of shrinking generator with controlling *p*-adic *FCSR* consists of the following steps:

1. All *LFSRs* from R_1 to R_{p-1} and *FCSR R* are clocked.

2. If the *p*-adic output $b_i = j$ of control register *R* is not equal to 0, the output bit of register R_j forms part of the keystream. Otherwise, if the output $b_i = 0$ of control register *R* is equal to 0, the all output bits are discarded.

In the origin papers of Klapper and Goresky only the case $p = 2$ is studied comprehensively [5]. It is shown [3], that the shrinking generator from Fig. 3 could use a generalization of *FCSRs* with stage contents and feedback coefficients in $\mathbf{Z}_p$ where *p* is a prime number, not necessarily 2. It is shown [5], that the work of the controlling *FCSR* depends on the initial loading of the register. Namely, the controlling *FCSR* ought to calculate the *p*-adic expression of a certain rational number $\alpha = (c/d) * p^e$, where *c*, *d* and *e* are integers, according to conditions in Eq. (3). Moreover, the initial loading of the *FCSR* register have to be:

$$a_0 + a_1.p + \ldots + a_{r-1}.p^{r-1} = \frac{c}{d} \bmod p^r . \qquad (9)$$

Here *r* is the number of cells tapped in the controlling *FCSR*.

It is not hard to see that usage of our class *p_adic*, proposed above, provides the finding of the necessary initial loading of the controlling *FCSR*.

From all the above stated and from [2], it is easy to see that the computation modeling of *p*-adic arithmetic, proposed in the paper, could be applied successfully for:

- developing of stream ciphers with *Feedback with Carry Shift Register* architecture;

- synthesis of pseudorandom sequences with variety of statistical properties such high linear span, low autocorrelation side-lobes and pair wise cross-correlation values, pair wise hamming distance.

ACKNOWLEDGEMENT

Authors would to thank Professor Carla Limongelli, who kindly give us opportunity to use her PhD thesis. We also acknowledge to Professor Rusin Petrov, whose observations made the drafting of the paper the best possible.

REFERENCES

[1] A. J. Baker, "An Introduction to *p*-adic Numbers and p-adic Analysis", Departament of Mathematics, University of Glasgow G12 8QW, Scotland, 2003.

[2] B. Y. Bedzhev, Zh. N. Tasheva, V. A. Mutkov, "CDMA Codes for the Next Generation Mobile Communication Systems", XII International Symposium of Theoretical Electrical Engineering ISTET 03, Warsaw, Poland, 6-9 July, 2003, Conference Proceedings, vol.I, pp. 78-82

[3] B. Y. Bedzhev, Zh. N. Tasheva, V. A. Mutkov, "An Shrinking Data Encryption Algorithm with *p*-adic Feedback with Carry Shift Register", XII International Symposium of Theoretical Electrical Engineering ISTET 03, Warsaw, Poland, 6-9 July, 2003, Conference Proceedings, vol.II, pp. 397-400.

[4] F. Q. Gouvea, "*P*-adic Numbers, an Introduction", Springer-Verlag, 1997.

[5] A. Klapper , M. Goresky, "2-adic shift register. Fast Software Encryption", Second International Workshop. (Lecture Notes in Computer Science, vol. 950, Springer Verlag, N. Y., 1994.) pp.174-178

[6] C. K. Koc, "A Tutorial on *p*-adic Arithmetic", Technical Report, Oregon State University, Corvallis, Oregon, USA, 2002.

[7] C. Limongelli, "The Integration of Symbolic and Numeric Computation by *p*-adic Construction Methods", Ph.D. thesis, University of Rome "La Sapienza", Italy, 1993.

[8] C. Limongelli, R. Pirastu, "Exact Solution of Linear Equation Systems over Rational Numbers by Parallel *p*-Adic Arithmetic", No.94-25, Johannes Kepler University, Linz, Austria, 1994.

[9] B. Schneier, "Applied Cryptography", John Wiley & Sons, New York, 1996.

[10] P. van Oorshot, A. Menezes, S. Vanstone, "Handbook of Applied Cryptography", CRC Press, 1996