**International Journal of Engineering Research and Development**
*e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com*
*Volume 10, Issue 3 (March 2014), PP.35-41*

# Hardware Algorithm for Variable Precision Multiplication on FPGA

S.Yazhinian[1], R.Marxim Garki [2]
[1,2]*Assistant Professor, Electronic and Communication Engineering,*
*Shri Krishnaa College of Engineering and Technology*

**Abstract:-** A hardwired algorithm for computing the variable precision multiplication is presented in this paper. The computation method is based on the use of a parallel multiplier of size m to compute the multiplication of two numbers of n×m bits. These numbers are represented in the variable precision floating point format, but in this work only the mantissas are considered; the exponents are easily obtained by adding the exponents of the two operands to be multiplied. In this computing method of multiplication, the partial products are added as soon as they are computed, resulting in the use of the lowest memory for intermediate results storage, (i.e. the size of the result is of 2n×m bits). The Xilinx FPGA circuits, of Vertex-II families and bigger ones, have interesting resources such as embedded multipliers 18×18 bits, memory blocks (Select Ram) and carry chain paths for the carry propagation acceleration and DCM blocks (Digital Clock Manager) to generate and control clocks. These resources have been advantageously used, in the implementation, to reduce the computation delay compared to the solution that uses only FPGA CLBs (Configurable Logic Blocks).
Our architecture has been tailored to use these efficient resources and the resulting architecture is dedicated to compute the multiplication of operands of sizes ranging from 1×64 bits to 64× 64 bits with a cycle time of 33 ns.

**Keywords:-** digital block manager, FPGA circuits, model sim PE, Intellectual Propriety, configurable logic block.

## I. INTRODUCTION

During the last decade, the computation speed of computers has increased dramatically. This is due to the development of VLSI technology that has enabled the integration of millions even billions of transistors on the same chip. This huge amount of transistors has let be possible the parallelism and the pipelining of operations at the hardware level to attain the current performances in terms of computation speed. However, the accuracy of these calculations has not been developed since 1985, appearance date of the IEEE-754 standard [1] which governs the floating point calculation. The computation power, offered by current processors, has led to the emergence of applications that are very consuming arithmetic operations. However the main problem of these computing capabilities is the accuracy of the results. In some applications, the errors can have dramatic consequences such as the case of software used in the transport means (air, rail, etc.), in the medical teams for patients (cancer) or in the army (defence systems etc.).They are parts of the list which is always longer by critical applications for the life or the security. An error in these applications can be very costly, as testifies the destruction of the Ariane 5 rocket during its first flight in 1996 [2] or can cost lives as confirms the failure of the antimissile Patriot during the first War of the golf in 1991 [3] or the overdoses radiation administered to patients in the National Oncology Institute of Panama in 2000 [4]. These errors are due, in great part, to the discrete nature of the floating point representation of numbers in the standard IEEE -754, where the numbers are represented on a fixed number of bits. Unfortunately, the accumulation of rounding errors and catastrophic cancellation can lead quickly to results that is completely inaccurate.The variable precision computing or "multi precision" [5], [6] allows to vary the computation accuracy according to the problem to be solved and the required precision for the results. To overcome the numerical limitations of actual processors, several programming languages, software dedicated libraries [7] as well as coprocessors [8] have been developed for the variable precision computing. This kind of computing is very useful when the problems to be solved are not very stable numerically or when a higher precision is required (i.e. greater than that available on computer). Nevertheless, these software solutions are slow and do not meet the requirements of applications that involve in plus of the accuracy of results, a high speed computation. To meet these requirements, a hardware implementation is inevitable. In this work, we exploit recent developments of FPGA circuits for (*Field Programmable Gate Array*) which offer re-programmability in addition of low cost development to design architecture for the fast computing of the multi recision multiplication of two large numbers with reduced hardware.The rest of this paper is structured as follows. Section 2 recalls the representation format of variable precision floating point numbers. Section 3 introduces the classical multiplication principle. Section 4 describes the Karatsuba multiplication. Section 5 presents the multiplication process of two large numbers which is the

core of our variable precision multiplier. The architecture of our multiplier isdetailed in section 6. Section 7 summarizes the implementation results and finally a conclusion is given in section 8.

## II. MULTIPLE PRECISION

Although the term "multiple precision" brings to mind applications such as determining π to billions of digits, most applications of high-precision arithmetic require only a few tens of digits, rather than hundreds or thousands. As an example, consider the Bessel function $J1(x)$ for large $|x|$ (up to 200 to 300). Of course, many subroutine libraries include $J1(x)$, but for many functions represented by similar formulas, no such libraries are available. For small values of $x$, we can use the convergent series (1) directly. This series is easy to understand and compute, but it becomes unstable for large $|x|$. For well-studied functions, such as $J1(x)$, algorithms suited for different ranges of arguments abound in the literature. For $J1(x)$, for example, there is a well-known asymptotic series in $|x|^{-1}$ that is stable for large $x$, as well as a backward recurrence relation that bridges the gap between the formulas for different ranges. A library routine for $J1(x)$ would include several different formulas and would select the best according to $|x|$. To do the same for less-commonplace functions might require mathematical research and extensive testing. But if we merely require a few (or a few thousand) values for small and moderate $|x|$, it might be more cost-effective to code just the power series, using multiple-precision arithmetic to control the instability. This brute-force method is often used to check library routines. In this article, we evaluate $J1(x)$ at $x = 35.3$ by summing the power series. Figure 1 uses Fortran 90 and 53-bit double precision on a 32-bit computer to sum the series. I have left the program in somewhat inefficient form so that it resembles the power series. Obviously, tuning the code for speed would entail replacing (–1)**K * X**(2*K+1) with TERM = –TERM * XSQ / (K*(K+1)). The program prints out the partial sums every few steps to exhibit the growing instability. Because the final result is about 15 orders of magnitude less than some of the partial sums, we can't have much confidence in any of that result's digits.

## III. VARIABLE PRECISION REPRESENTATION OF NUMBERS IN FLOATING POINT ARITHMETIC

In variable precision arithmetic, two representation formats of numbers are used: the fixed-point representation and the floating point representation. This later was chosen to be used in our application as it allows representing more numbers and has larger dynamic than the fixed-point representation. This format is shown on figure 1. It consists of an exponent (E), a bit sign (S), a type (T), a length of the mantissa (L) and a mantissa (M) which includes (L+1) words (M(0) to M(L)). The exponent has a fixed length and is represented in 2'complement format. The sign bit is equal to zero if the number is positive and equal to one if the number is negative. The type indicates whether the number is infinite, zero or not a number. The length specifies the number of m bits words in the mantissa. The words of the mantissa are stored from least significant word M (0) to the most significant word M (L). The mantissa is normalized between 1/2 and 1.

## IV. THE CLASSICAL MULTIPLICATION

The simplest multiplication algorithm is the one used when we do a multiplication "by hand": multiply each digit of one operand by every digit of the other operand then do the appropriate shifts and finally add all the partial products.

In this "manual" method, we begin by computing all the partial products before adding. In the "computer" version, additions are made progressively to avoid the unnecessary memorization of the n numbers of (n+1) digits. Contrary, if we do not have an n×m size multiplier (for large operands), the complexity will increase as the multiplication of the operand by a digit cannot be performed in one operation. In this case, the operands are subdivided into several words which sizes are equal to the data path size which is equal to m bits, then the multiplication is done according to the concept illustrated on the **figure 2.**

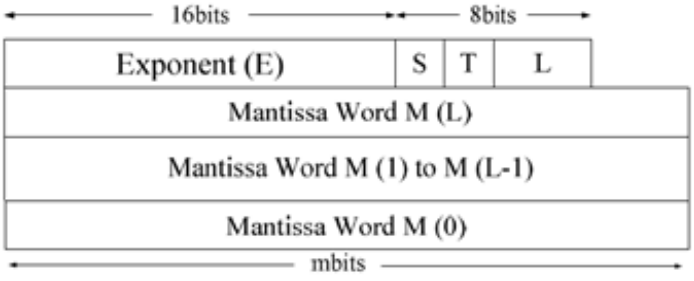


Figure 1. Variable precision floating point representation

In this method, the operands size is supposed equal to n digits of m bits and the computation of a partial product, which is the multiplication of one operand by a digit of the other operand, requires n multiplications of (m×m) bits and (n-1) additions of m bits. Hence the total number of operations, to carry out a multiplication of two operands of (n×m) bits is equal to $n^2$ multiplications of (m×m) bits and [n×(n-1)+2n] additions of m bits.

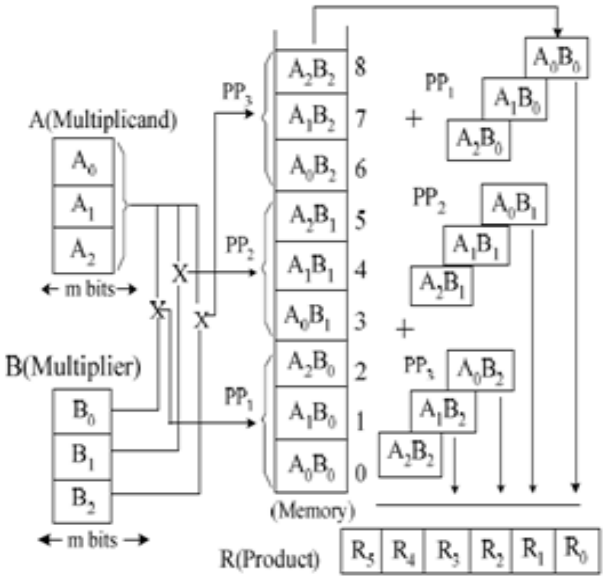


Figure 2. The classical multi precision multiplication of two numbers of 3 words of m digits.

Nevertheless, this multiplication method, which consists in computing the partial products then storing them in a memory and after do the final addition, is very costly in terms of memory, since all the partial products must be stored. For a multiplication of two numbers of n digits of m bits, the memory required to store all the partial products is $n^2 \times (m+1)$ bits. For numbers represented on 512×64 bits length, the memory required to store all the partial products is about 17 Mega bits.

## V. KARATSUBA MULTIPLICATION

The Karatsuba algorithm is a recursive algorithm introduced by two Russian mathematicians Karatsuba and of man in 1962. This algorithm is based on the splitting of the multiplier and the multiplicand into two parts: the least significant $A_L$ and $B_L$ and the most significant AH and BH.

| | | | | |
|---|---|---|---|---|
| A = AL + 2n/2 AH, B = BL+2n/2BH, Then the product is as follows: A×B=AHBH2n+(AHBL+ALBH)2n/2+A LBL | | (1) | | 1 |

As we can see this method needs 4×n/2-bit multiplications and 3×n-bit additions.

In 1963 A. Karatsuba and Y. Ofman described a divide and conquer multiplication algorithm [9]. Using their algorithm, n-bit multiplications are divided into n/2-bit multiplications by the following equation:

| | |
|---|---|
| □ _×_ □ =AHBH(2ת2n/2)+(AH+AL)(BH+BL)2n/2+ALBL(1-2n/2) | (2) |

This method needs only 3×n/2-bit multiplications, but 2×n-bit additions and 4×n/2-bit Additions. It is clear that the Karatsuba algorithm [10] is based on the substitution of a Multiplication by two additions, since it reduces the number of multiplications by one at the expense of two additions. Hence the performance of this algorithm is based on the performance gap between the two operators (multiplication and addition) used to achieve this algorithm. It was reported in [11], that the complexity in terms of execution delay of n-bit multiplication by Karatsuba algorithm is $T(n) = O(n^{\log 3})$. This last is well lower than the complexity of the classical algorithm which is $T(n)= O(n^2)$.The Karatsuba multiplication is quite often implemented in software for variable precision computing. This algorithm is more efficient compared to the classical algorithm when the operands size exceeds a given threshold. This threshold is often an experimentation result and depends on the computer used and how this method is computed [12]. Despite the low algorithmic complexity degree of Karatsuba algorithm, its hardware implementation is often complex and depends on the implementation devices

(ASIC, FPGA, etc.).For an FPGA implementation, the Karatsuba multiplication presents a complex routing which increases with the operand's size, due to the successive divisions of operands to obtain sub words whose size is equal to the multiplier size used in the architecture. This routing complexity is more important if the ratio: operand's size/multiplier size is larger.

## VI. THE PROPOSED METHOD

The proposed method presented in this section is simply based on the classical multiplication method with solution to the drawback of using large memory. In this method, a memory of only (n×2m) bits is used instead of ($n^2$x2m) bits memory. This reduction is more important when the operands size is bigger. A particular interest was granted to adapt this method to the Xilinx FPGA circuits, which contain interesting resources for the multiplication implementation of large numbers.

Consider the multiplication R=A×B where:

$$A = \sum_{i=0}^{n-1} A_i \times 2^{i \times m}, \qquad B = \sum_{j=0}^{n-1} B_j \times 2^{j \times m}$$

$$R = \sum_{k=0}^{2n-1} R_k \times 2^{k \times m} = \left(\sum_{i=0}^{n-1} A_i \times 2^{i \times m}\right) \times \left(\sum_{j=0}^{n-1} B_j \times 2^{j \times m}\right)$$
$$= \sum_{j=0}^{n-1} \left(\sum_{i=0}^{n-1} (A_i \times B_j)\, 2^{(i+j) \times m}\right)$$

As shown on figure 3, operands A, B and the result R are split into m-bits words that correspond to the size of both the multiplier and the memory cells used in this method.

In the previous section, we talked about the routing complexity generated in the implementation of the Karatsuba method and the importance of the routing delay in the implementation of complex functions on FPGA circuits.

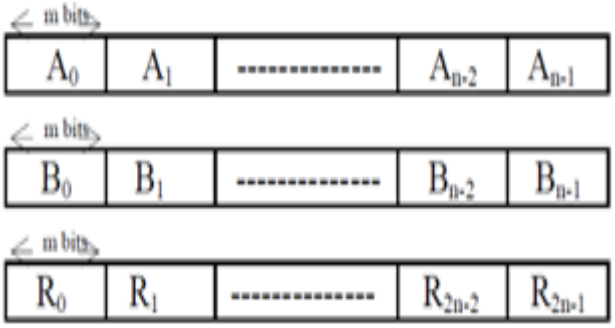


Figure 3. Operands splitting in m-bits words

To reduce this routing complexity, which sometimes cause additional delays more important than those of the logic? Our method is based on the accumulation of products $A_i$□ $B_j$ as soon as they are computed. The algorithm for calculating the variable precision multiplication is given as follows:

**Variable precision Multiplication Algorithm**

```
Inputs  A, B
Output  R= A×B
Initialization R= 0
        for j=0 to n-1 do
            for i=0 to n-1 do
            R(i+j) = LSB de (Ai×Bj) + R(i+j) ;
            R(i+j+1) = MSB de (Ai×Bj) + R (i+j+1) ;
            end for
        end for
Return R = (R0, R1,…., R2n-1)
```

An example for computing this multiplication of two numbers of 3m bits is illustrated on the figure 4

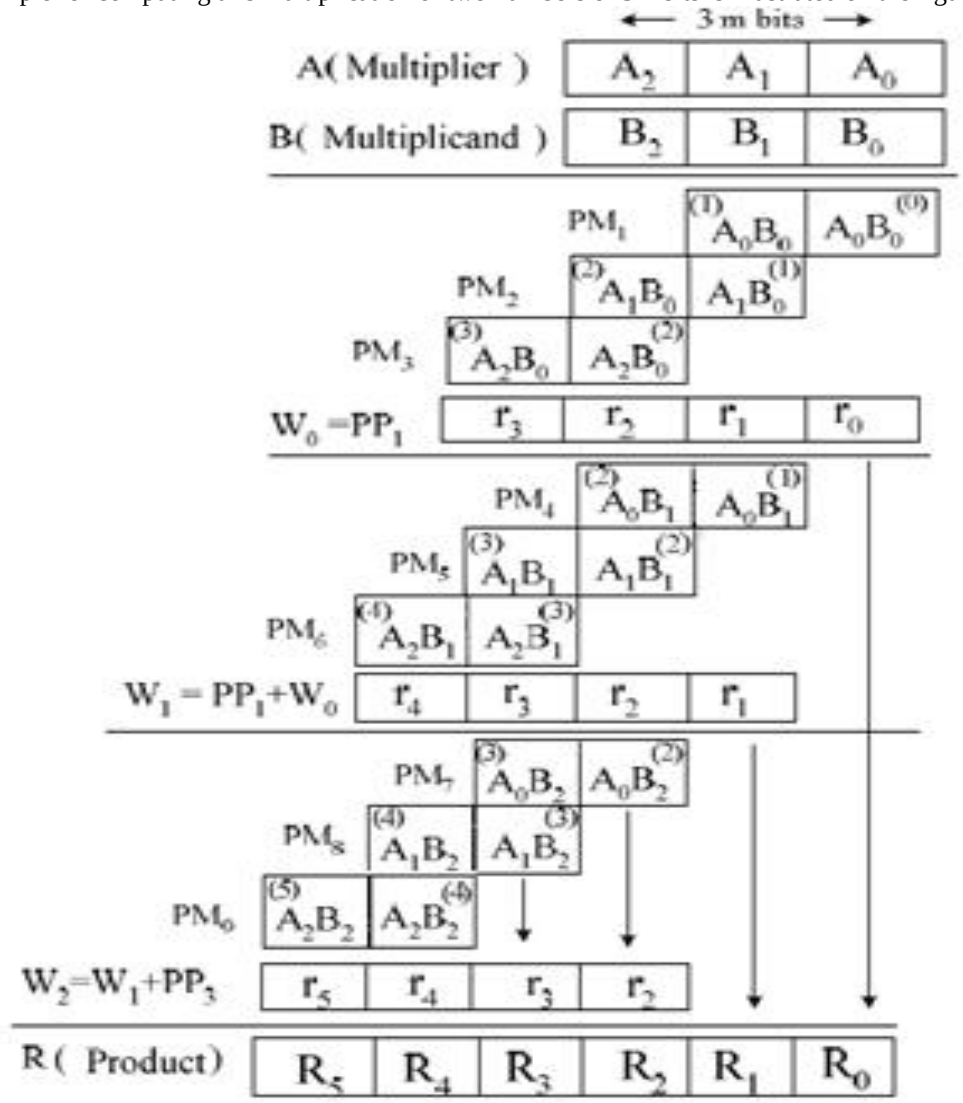


Figure 4. The proposed process for computing the multi precision multiplication

## VII. ARCHITECTURE

The architecture implementing the method described in the previous section is presented on the figure 5. In this architecture, the operands A and B are first stored in two memories MA and MB of n words of m bits. Each word Ai, Bj of operands A and B is addressed by its weight respectively i and j, which represents their position in the MA and MB memories.

Each iteration of the product $A_i \times B_j$ is performed by the parallel multiplier 64 × 64 bits and the result is on 128 bits. The 64 least significant bit (LSB) have a weight of (i+j) while the 64 most significant bits (MSB) have a weight of (i+j+1).

The LSB and MSB of the multiplication result are added to the results of the preceding iteration stored in the memory result MR respectively at the addresses (i+j) and (i+j+1). The results of these two additions are stored once again in the same addresses. This process continues until the last product $A_{n-1} \times B_{n-1}$.

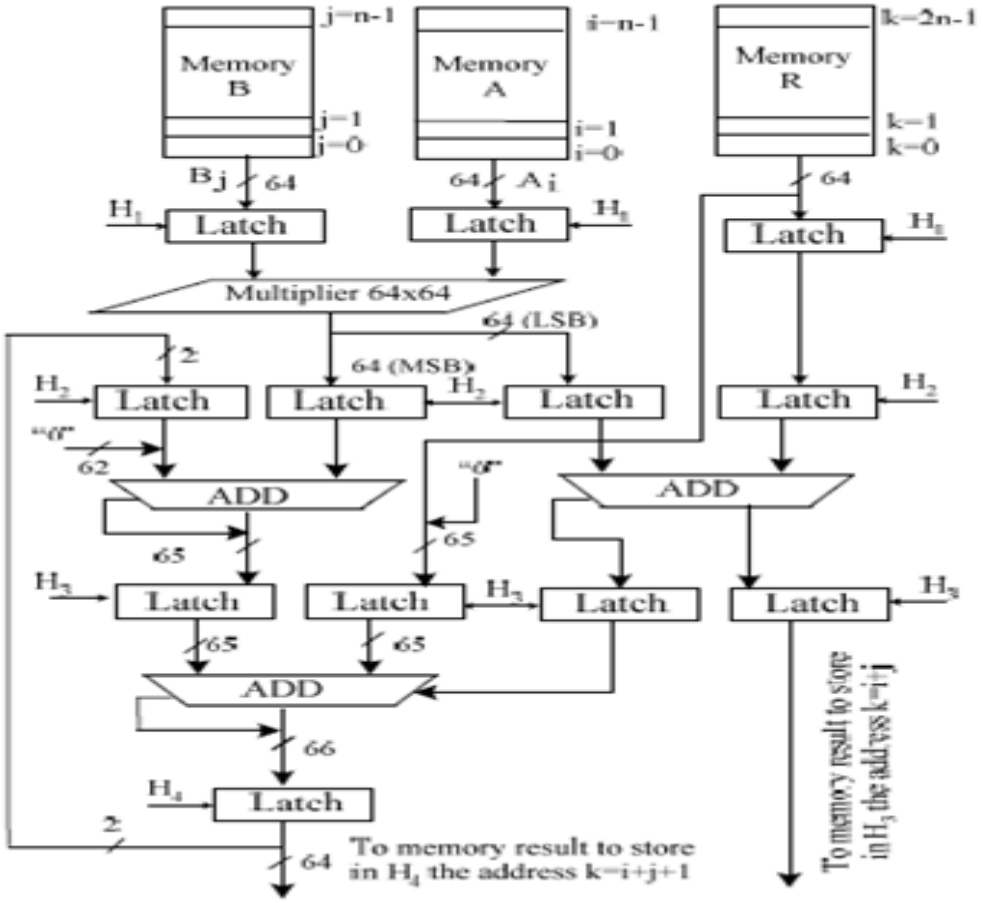


**Figure 5. Architecture computing the multi precision multiplication**

## VIII. IMPLEMENTATION RESULTS

Our architecture has been implemented using Foundation series 7.1 of Xilinx environment. All modules constituting the architecture have been generated by using the CORE generator system. To guarantee the correct behaviour of our architecture; this last has been simulated using Model Sim PE 6.0, then synthesized employing the XST tool of Xilinx. It has been mapped placed and routed on the Xilinx FPGA circuit of virtex-2 family, the XC2V1000 (-5) bg575.The implementation results of this architecture are presented in the table 1.

TABLE I. OCCUPATION RATE OF THE ARCHITECTURE ON THE XC2V1000 FPGA CIRCUIT.

| Element | Occupation rate | % |
|---|---|---|
| IOB | 72/392 | 21% |
| SRAM | 3/48 | 7% |
| Slices | 2381/7680 | 11% |
| Mult 18x18 | 16/40 | 40% |
| BUFGMUX | 11/16 | 68% |
| DCM | 3/8 | 37% |

## IX. CONCLUSION

In this paper, a hardwired method for computing the variable precision multiplication was presented. It took its advantages of the classical multiplication method which presents a low routing complexity compared to Karatsuba multiplication with judicious use of resources provided by FPGAs circuits to implement an architecture offering a delay of $n^2 \times 33$ ns (for the multiplication of two numbers of n×64 bits). This multiplier is suitable to be used as IP (*Intellectual Propriety)* in an embedded system for applications requiring the multi precision computing. The operand sizes, supported by our architecture, are ranging from (64x1) bits to (64×64) bits. Nevertheless, these sizes can be bigger insofar as we have used only 7% of SRAM resources (Table 1) and that changing the length of the memory (i.e. n) will have influence neither on the architecture nor on its performance.

## REFERENCES

[1]. M.Daumas, F.Dinechin, A, Tisserand, "L'arithmétique des ordinateurs", <Réseaux et systèmes répartis>- Calculateurs parallèles, Volume 13 n°4-5/2001.
[2]. Douglas N. Arnold, "The Explosion of the Ariane 5", http://www.ima.umn.edu/~arnold/disasters/ariane.html, 2000.
[3]. Douglas N. Arnold, "The Patriot Missile Failure", http://www.ima.umn.edu/~arnold/disasters/patriot.html , 2000.
[4]. WISE News Communique "Radiological accident in Panama", http://www10.antenna.nl/wise/index.html?http://www10.antenna.nl/wise/549 /5278.html, June 2001.
[5]. J.-C Bajard. ; L. Imbert; F. Rico, "Evaluation rapide des fonctions élémentaires en multi précision", TSI : Technique et Science Informatiques, , Vol. 20, n° 2, pp. 267-286, 2001.
[6]. M.Quercia, "Calcul multi précision", http://pauillac.inria.fr/~quercia/papers/multiprecision.ps.gz, 2004.
[7]. D.M.Smith, "Using Multiple Precision Arithmetic", Computing in Science &Engineering IEEE publication, July/august 2003.