Design of a Multiplier for Similar Base Numbers Without Converting Base Using a Data Oriented Memory

Majid Jafari[*], Ali Broumandnia, Navid Habibi, Shahab Forgani

Department of Computer, South Tehran Branch, University of Islamic Azad, Iran

Abstract

One the challenging in hardware performance is to designing a high speed calculating unit. The higher of calculations speeds in a computer system will be pointed out in terms of performance. As a result, designing a high speed calculating unit is of utmost importance. In this paper, we start design whit this knowledge that one multiplier made of several adder and one divider made of several sub tractor. Therefore, if the fast adder or fast multiplier designed, performance will be improved. In this design, a circuit is designed in a manner that without a need for transforming numbers or letters from the given bases into binary bases, the multiplication of two numbers for the same base is done in that base. Reduction in the number of conversions in the calculating unit, causes reduction in the consumption power and an increase in the operating speed of the system. In this design, a very small Data Oriented Memory is used to save numerical and character data.

Keywords: Data Oriented Memory; same base multiplication; Content Addressable Memory; Sub tractor; Category.

1. Introduction

Each computer system has four chief units including control, memory, function unit and I/O units. In this architecture, the memory unit which is data oriented memory type with reading and writing capabilities is used to store data type of numbers and ASCII. The control unit that control word bit and select line multiplexers in data path, fetches instructions from memory and performs the operations sequentially [2]. The function unit performs the computational operations. Each information system requires a processor to process the instructions.CPU is responsible to perform this task in computers. CPU processes almost all instructions given by software and hardware by using logical operations, mathematical computations and comparisons. This unit computes and compares all input instructions using ALU or decide based on logical operations and then returns output, if necessary. This process is done over the processor's registers as the CPU's work desk. Mathematical operations mean a number of simple operations such as multiply, division, sum and subtract. The increasing speed of the whole system is determined according to increasing the speed of the slowest part of system, namely ALU. The higher the speed of operations in this unit, the faster processing the CPU would have and if CPU has high speed in processing,

* Corresponding author. Email: Iran.jafarymajid@yahoo.com

the system would have better speed and efficiency and lower throughput.

The memory to be useful in this article is a data-oriented memory (DOM) which has been designed for data-oriented models according to data-oriented theory. This theory introduced in recent years uses a type of data structure, called problem – solution data structure to solve problems [3]. Data-oriented theory is based on this fact that if enough and proper data are used to model the principles, the complexity of the required algorithms to solve the problem is decreased and ratio the time required to solve the problem is decreased. The applied memories to data-oriented models are a certain type of content addressable memories(CAM)where they use the data itself instead of address to search a value. So, they would have faster in order to access data. Although data oriented memories are considered as a certain type of CAM, but they are different in terms of performance. The CAM's examine only whether the key exists or not, but DOM's make the desired answer available if the applied key is matched [2-4].

The above mentioned model and architectures have been designed to work with regular data for integers and ASCII codes in same base multiplier's architecture.

2. Related works

Among different data processing in digital computers, we could point out those functions related to various types of arithmetic operations. The main arithmetical operation is to sum up two binary digits. We know that one multiplier made of several adders and one divider made of several sub tractor. IF simple sum in cludes four basic operations, so simple multiplication also includes four basic, 0 * 0 = 0, 0 * 1 = 0, 1 * 0 = 0 and 1 * 1 = 1. The three first operations generate 0 digit, but when both Multiplicative bits are 1, the binary multiplication generate 1 digit.

2.1 Types of adders

Multiplier plays an very important role in today's digital circuits. The design of high speed, low power consumption, less area, and low irregularity in layout are very important. There are various types of multipliers concluding Twin-Piped Serial-Parallel Multiplier, Array Multiplier, Wallace Tree Multiplier, Modified Booth Multiplier, and Combined Modified Booth-Wallace Tree Multiplier.

2.1.1 Binary multiplier

In grammar school we learned to multiply by adding a list of shifted multiplicands computed according to the digits of the multiplier. The same method can be used to obtain the product of two unsigned binary numbers. Forming the shifted multiplicands is trivial in binary multiplication, since the only possible values of the multiplier digits are 0 and 1. An example is shown below:

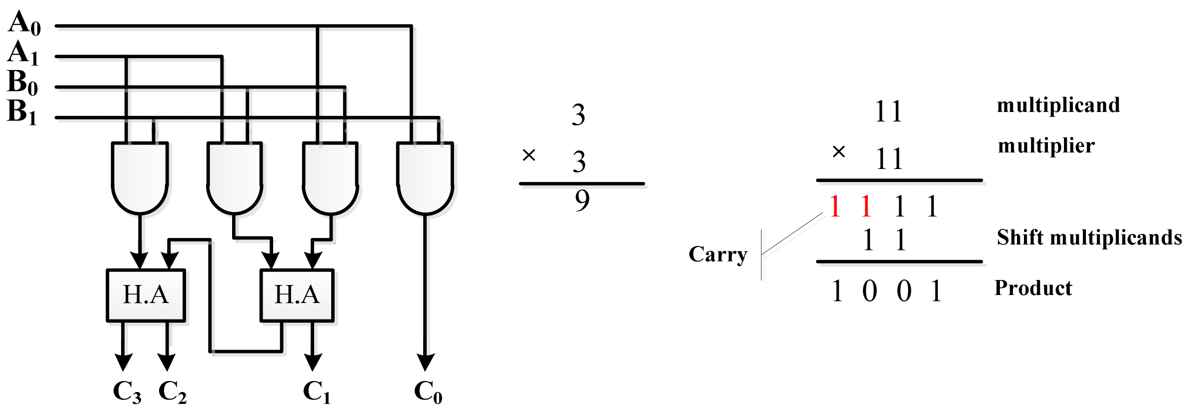

Fig. 1. Two numbers couplet multiplication with example

In general, when we multiply an n-bit number by an m-bit number, the resulting product requires at most n + m bits to express. The shift-and-add algorithm requires m partial products and additions to obtain the result, but the first addition is trivial, since the first partial product is zero. Although the first partial product has only n significant bits, after each addition step the partial product gains one more significant bit, since each addition may produce a carry.

bits are B1 and B0, the multiplier bits are A1 and A0, and the product is C3C2C1C0. The first partial product is formed by multiplying B1B0 by A0. The multiplication of two bits such as A0 and B0 produces a 1 if both bits are 1; otherwise, it produces a 0. This is identical to an AND operation. Therefore, the partial product can be implemented with AND gates as shown in the diagram. The second partial product is formed by multiplyingB1B0 by A1 and shifting one position to the left. The two partial products are added with two half-adder (HA) circuits. Usually, there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products. Note that the least significant bit of the product does not have to go through an adder, since it is formed by the output of the first AND gate. A combinational circuit binary multiplier with more bits can be constructed in a similar fashion. A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier. The binary output in each level of AND gates is added with the partial product of the previous level to form a new partial product. The last level produces the product. For J multiplier bits and K multiplicand bits, we need (J* K) AND gates and (J –1)K -bit adders to produce a product of (J + K)bits.

2.1.1 Binary parallel multiplier

In this section we outlined an algorithm that uses n shifts and adds to multiply n-bit binary numbers. Although the shift-and-add algorithm emulates the way that we do paper-and-pencil multiplication of decimal numbers, there is nothing inherently “sequential” or “time dependent” about multiplication. That is, given two n-bit input words A and B, it is possible to write a truth table that expresses the 2n-bit product P = A.B as a combinational function of Many approaches to combinational multiplication are based on the paper-and-pencil shift-and-add algorithm. Figure 2 illustrates the basic idea for an 5 × 5 multiplier for two unsigned integers, multiplicand A =a4 a3a2a1a0 and multiplier B= b4b3b2b1b0 we call each row a product component, a shifted multiplicand that is multiplied by 0 or 1 depending on the corresponding multiplier bit. Each small box represents one product-component bit ypc: the logical AND of multiplier bit Bi and multiplicand bit Aj. The product P = P5P3P2P1P0has 6 bits and is obtained by adding together all the product components. A and B. A combinational multiplier is a logic circuit with such a truth table.

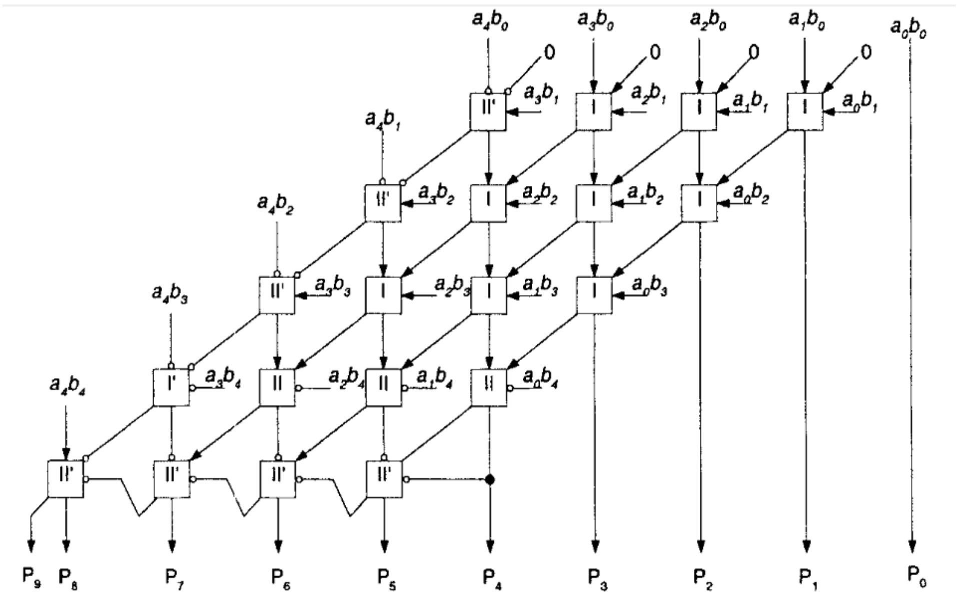

Fig. 2. 5-by-5 Pezar is Array Multiplier [50]

product-component bits have been spread out to make space, and each “I” box is a full adder. The carries in each row of full adders are connected to make an5-bit ripple adder. Thus, the first ripple adder combines the first two product components to produce the first partial product, as defined in Subsequent adders combine each partial product with the next product component.

It is interesting to study the propagation delay of the circuit. In the worst case, the inputs to the least significant adder (A_1B_0 and A_0B_1) can affect the MSB of the product (p5). If we assume for simplicity that the delays from any input to any output of a full adder are equal, say t_{pd}, then the worst-case path goes through 20 adders and its delay is $20f_{pd}$. If the delays are different, then the answer depends on the relative delays.

3. Disadvantages of the designed multiplier

The designed circuits have the following major disadvantage:

The above circuits have been designed only for binary computations. For non-binary and other bases (for example k base), at first that base must be converted into base 10, and then it must be converted to base 2. After obtaining the result, it is converted to base 10 and then to the given base, where in this case, to obtain the multiplier and divider of two numbers in base 10 and non-10 and binary bases, base conversion is required two times and four times, respectively. Also, the functional unit requires 10, 40, 2, 3 and 20 hour cycles for each multiply, divide, add, subtract and exponent operations in normal design but in advanced design this times is decreased to 5, 15, 2, 3 and 10 [4]. The required time to convert other bases to base 10 is equal to the number of digits multiply by 10, for exponent 20 and sum of them 2hour cycles in normal design (where it is required to multiply each digit in the given base by the exponent of the digit place). Now, the obtained result must be taken in to base 2which is obtained by successive divisions of the obtained decimal number by 2. The time complexity of each division is 40 hour cycle. So, the required time to convert a decimal number in two base 2 is equal to the number of division by 40 cycles. After addition operations (which requires 2 hour cycles) and obtaining the addition result by the mentioned adders, again we would have the conversion operations from base 2 to base 10 and from the base 10 to the given base. This stages Is shown in Figure 3.Therefore, the latency of multiplying two same base non binary numbers by mentioned adders and Multiplications is high in usual case. As a result, in section 4, a circuit has been designed that does not require any conversion and it is able to compute the multiplier of two same base numbers quickly by using a data-oriented memory without base conversion.

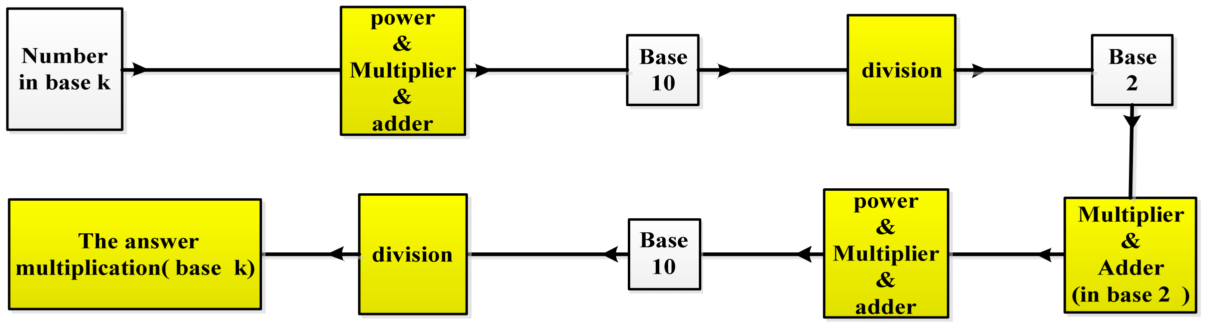
Number in base k
power & Multiplier & adder
Base 10
division
Base 2
Multiplier & Adder (in base 2)
power & Multiplier & adder
Base 10
division
The answer multiplication(base k)

Fig. 3. Stages of multiplication two numbers in k base

4. Introducing the circuit of the new multiplier and memory

In this design, a sequential circuit with inputs (multiplier and divider)connected to the data oriented memory is considered. The memory has been embedded in the input and output of the ordinal circuit in order to take the input and return the result (as binary or ASCII code for hex base) without conversion to accelerate the operations. The new memory being introduced is to manage data with numerical and letter types. This memory is a data-oriented memory in which numbers from 0 to 256 and letters A-F are stored in memory with their binary values, so that as soon as a number or letter is entered their binary values become available without conversion. The base of these memories is content addressable memories [2-4]. Although data-oriented memories are a special type of content addressable memories, but they are different in terms of performance. The content addressable memories just examine the existence of the key, but data-oriented memories make the given result available if the applied key is matched. In this type of memories a part of data is considered as an address. When a data is searched, a part of data is compared with the given data instead of address. If matched, then the restof values are entered into bus [3-6]. A key in a binary format is considered which is compared with all memory cells in a parallel manner so that if there is a match, two successive cells where the problem is placed among them are selected. To perform matching, at first the applied key is compared with all the problems in memory bit by bit [7-8]. As a result, we could conclude the equality of two corresponding bits by the Boolean function in equation (1):

$$a_i = x_i.b_{j,i} + x_i'.b_{j,i} \qquad (1)$$

$$Match_j = \prod_{i=1}^{i=n-1} a_i \qquad (2)$$

$$H_{it} = \sum_{j=0}^{j=k} (a_0.\prod_{i=1}^{i=n-1} a_i)\,j = \sum_{j=0}^{j=k} (a_0.Match_0)\,j \qquad (3)$$

Where index *j* representsa memory cell and index *i*is the bit number in each cell. By enabling the matching signal obtained from equation (2), the results corresponding to the enabled cells are placed in the buses as the sub results of 1 and 2 andthe processing unit sends the final result to the output by using the result within the bus. In equations (2) and (3), j represents the memory place and i is the bit number relates to the memory cells. In this design, the input unit has a keyboard and data-oriented memory where they are compared in parallel by each pressing the key of the comparison unit within the data-oriented memory (control circuit) [4-5]. This memory has consisted of two parts where in the first part, ASCII codes and in the other part the binary values related to ASCII codes have been embedded. Once the searching value matches with the ASCII code of a memory cell, the ASCII and binary codes relating the above cell are placed in two buses 1 and 2where they could be used based on the requirement (in this design, the ASCII code enters into bus 2 has not been used) [6-7]. The inputs related to the arithmetic unit are connected to bus 1. So, when some values of keyboard are entered into the input of the adder to add two numbers, their binary code is placed within the bus and the arithmetic unit starts to add two binary numbers in the desired base.

The data-oriented memory in the designed circuit is as follows. XNOR gates have been used to comparison. When a key is pressed from the keyboard, its corresponding ASCII code is searched in the memory and the binary equivalent of the selected key's ASCII code is placed in bus 1 and the ASCII code's value itself is placed in bus 2. Since an arithmetic unit works with base two in the functional unit, the input values of arithmetic unit (A_i, $B_i$andCi) are provided by bus 1. When the sum of two values wantsto be sent to the screen, bus 2 is used. It means that the obtained value sends the sum result to the control circuit as an ASCII code and the control circuit sends the ASCII code equivalent of that number to the screen.

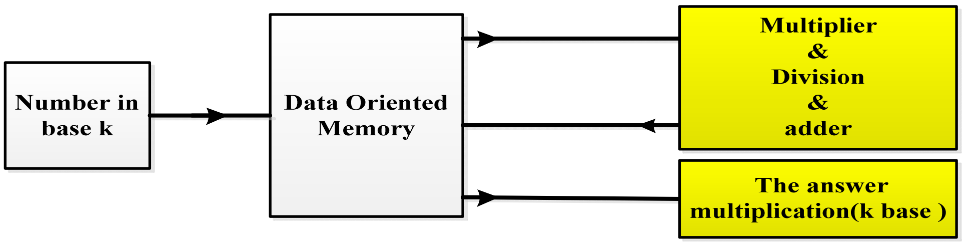
Number in base k
Data Oriented Memory
Multiplier & Division & adder
The answer multiplication(k base)

Fig. 4. Stages of multiplication two numbers in k base whit data oriented memory

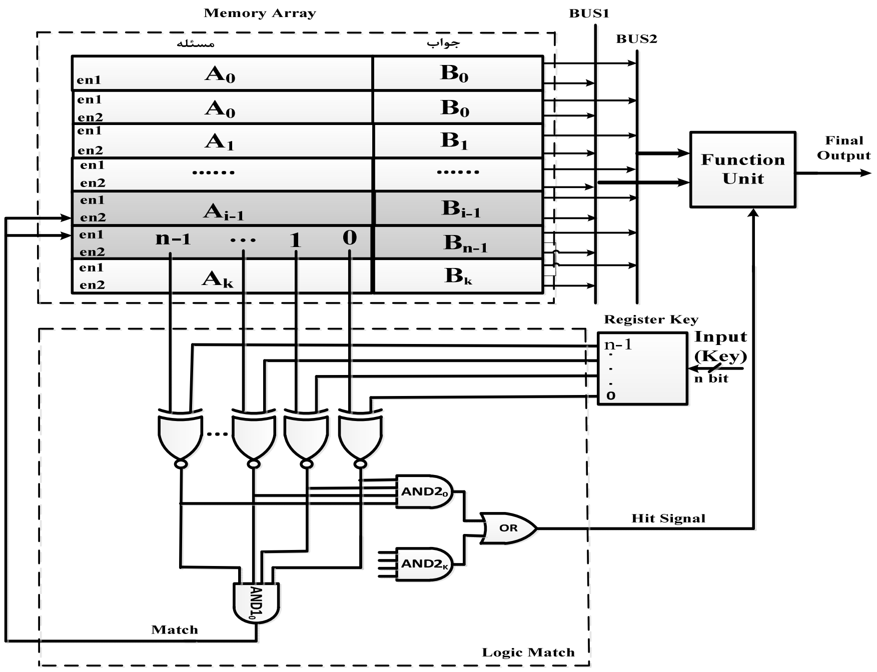

Fig. 5. data oriented memory architecture [51]

In this memory A shown ASCII and B shown binary codes. When decimal numbers are applied as input (key) to the memory, the ASCII code of the values is searched, if matched the binary values of that cell is placed in bus 1. The function unit that including shifter and alu, uses the binary values within bus 1 as the values of inputs register A_x, B_x and C_x of function unit. It then applies the binary values of the output obtained from the functional unit to the memory as a key, if matched; the ASCII code of the matched cell is placed in bus 2 in order to be shown in the output. It is worth noting that the control unit uses data-oriented memory for each inputs A_x, B_x and C_x separately. It means than when the input A_x is decoded by decoder and receives its values from the memory, the control unit clears the bus and the decoder starts to decode the input B_x. Input C_x is either zero or one. This input is also determined by control unit which becomes zero or one depending on addition or subtraction. An example is given to better understand the performance of the data-oriented memory. In this example, key 9 = $(00001001)_2$ has been pressed from the keyboard. The system applies its ASCII equivalent to the data-oriented memory in order to convert to base two. To perform the matching action, at first the applied key in binary format is compared as bit by bit with all the problems in memory. It means that by applying the ASCII code equivalent of letter 9 as a problem, the key is compared with all memory cells in order to find a sub-range including the key by XNOR gates. According to figure 6, the matching is occurred only in the A_9 cell of memory and the output of *And* A_9 becomes 1, then signals *en1* and *en2* are activated and two memory cells, A_9 cell relating to ASCII code A_9 and a cell relating to the binary corresponding to that ASCII code, B_9, are placed in the bus as the sub results of 1 and 2. In this case, where a matching between the applied key and one of the problems in the memory has occurred, the hit signal is activated and the arithmetic unit receives its inputs by using sub results and according to the inputs sends the final result to the output. Also, when the final result is returned from the arithmetic unit to be represented, this value is applied as a key to the data-oriented memory so that its ASCII equivalent is determined in the output to be shown [6-7].

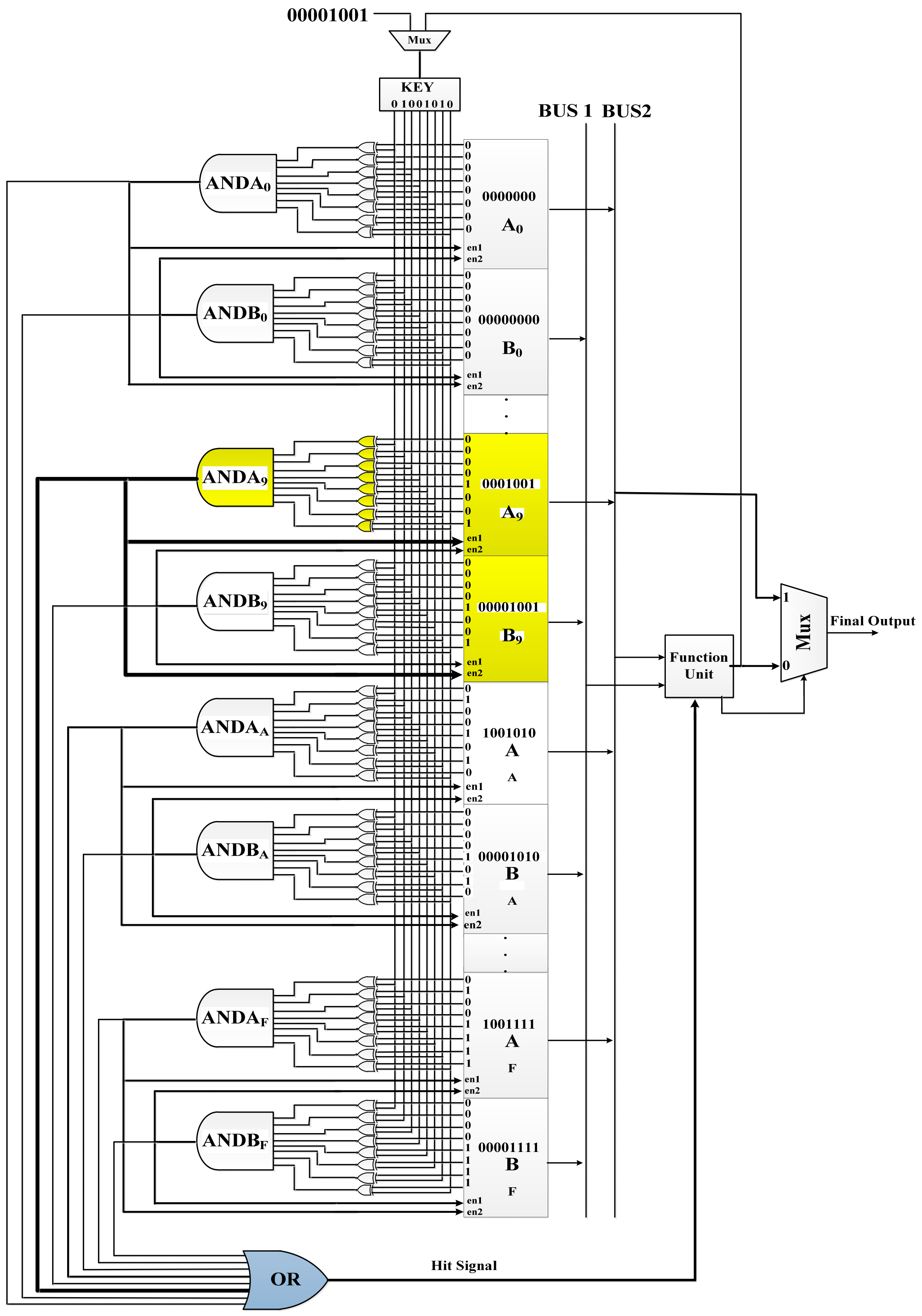

Fig. 6. An example of a data-oriented architecture

4.1 Design a new arithmetic unit

To design a new multiplier, the way of mathematical computing is important which is described in the following. In this design we have need to data oriented memory, multiplier-divider and same base adder.

4.1.1 Multiplication of two numbers in the same base

The way of adding two numbers that has been achieved from multiplier in same base is as follows. Numbers were obtained include $T_0T_1\ldots T_2T_{n-1}$levels (level means the position of digits such as one, decimal, hundreds and …), when the multiply of two numbers in level $T_0$is lower than the base, the obtained multiplication of among multiplications value will be the multiply of two numbers in level T_0. But when the multiplications of two numbers in level T_0 is greater than or equal to the base, the obtained multiplication value will not be the multiplication of two numbers in level T_0. Since the values in base must be at most one unit less than the base (for example, in decimal number A= $a_4a_3a_2a_1a_0$, as the base is 8, so $a_i<8$), so at first the multiplication and sum of them of two numbers must be subtracted from base, the remainder of subtraction (division) should be the multiplication of two numbers in level $T_0$and a carry digit(The base coefficient) is shifted to level T_1.And this is continued until level n. With that in mind $A_i = B\,k_i + R_i$ that B is base, k is carry for next Category and R (that is remains) is the answer of multiplied in this category to multiplication the new method We have the two following relations:

The first method:

In this way, at any stage of multiplication referred into division and data oriented memory unit. After the refer and Response to the multiplication result (change to $P_f= A_i\,.B_j$ position) in this category, the carrier (that obtained of division quotient $L_i(P_f = K\,L_i + R_i$)) will be added to the next category and the remaining would be multiplied answers. Then the first stage begins again.

$$
\begin{aligned}
&P_f = A_i\,.\,B_j \qquad , \quad i\,,j\,,f = 0,1,2,3,\ldots\\
&P_f = K\,L_i + R_i\\
&(P_{f+1} + L_i\,)= KL_{i+1} + R_{i+1}\\
&(P_{f+2} + L_{i+1}\,)= KL_{i+2} + R_{i+2}\\
&\cdot\\
&\cdot\\
&\cdot\\
&(P_n + L_{n-1}\,)= KL_n+ R_n
\end{aligned}
$$

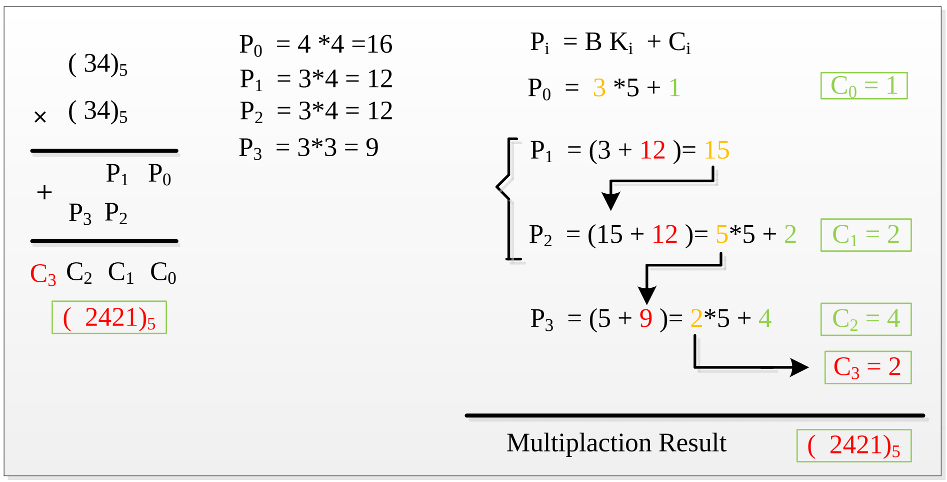

Fig. 7. An example of multiplication

The second method:

Due to increase the delay due to many references to the memory unit and divider unit, the previous method was not suitable, so other way the we recommend.

In this method we are using a stack to store values temporarily, The results of temporary is called V_i. The results of multiplication of previous steps and stored in stacks were called P_f. To obtain the mid result, its necessary to obtain all of addition (in normal form) P_f that are in a same category.

$$
\begin{aligned}
&P_f = A_i . B_j \quad , \quad i , j , f = 0,1,2,3,\ldots \\
&V_f = \square P_f \\
&P_f = K L_i + R_i \\
&(V_{f+1} + L_i) = KL_{i+1} + R_{i+1} \\
&(V_{f+2} + L_{i+1}) = KL_{i+2} + R_{i+2} \\
&. \\
&. \\
&. \\
&(V_n + L_{n-1}) = KL_n + R_n
\end{aligned}
$$

In figure 8, the multiply of two numbers 34 and 34 has been shown in base 5.

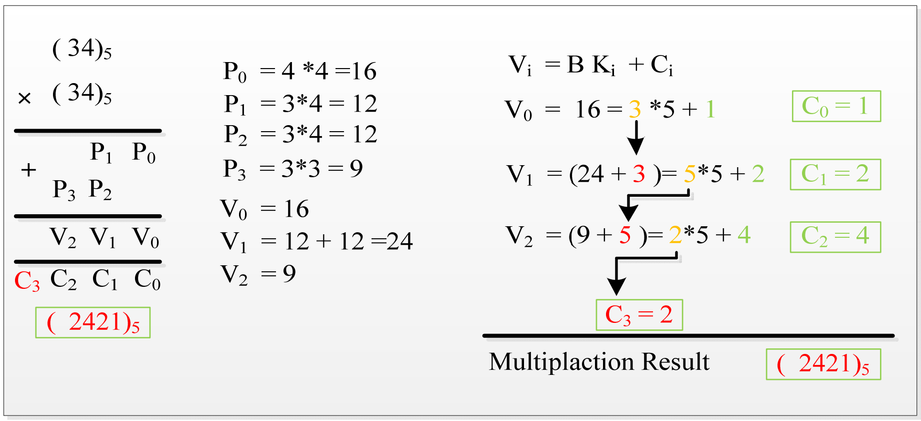

Fig. 8. An example of multiplication

In figure 8, the multiply of two numbers 34 and 34 has been shown in base 5, as it can be seen, numbers 4 and 4 are multiplied in the first level where the result is 16, but this is not the result in the first level, because the result would be more than base 5, so 16 is subtracted (subtraction is k times that k carry for next category (division)) from base 5, the result is 1, then the multiply of 3 and 4 is placed in the first level and a carry digit is shifted to the second level to be added. In the second level, the carry digit 3 is summed up with the sum of multiply(4 * 3) + (3*4) and the result becomes 27, since the result is greater than the base, so 27 is subtracted from base 5, the result is 2 and carry is 5 (27 = 5*5 + 2), then the multiply of 5with (4 * 3) + (3*4)is placed in the second level and a carry digit is shifted to the third level to be added. In the third level, (3 * 3) and the carry number are summed up where the result is 4, because the result is lower than the base, therefore 4 is placed in the third level and 2 carry for four level.

The architecture of the same base adder is shown in figure 9 based on the way of adding two same base numbers. In this design, a mentioned arithmetic logic has been used to add two same base numbers.

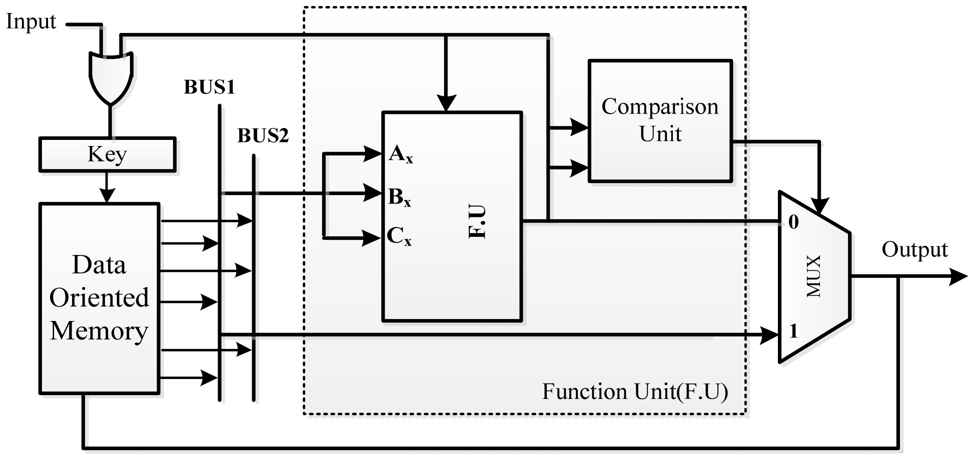

Fig. 9.General architecture of same base adder [51]

In the above architecture, two same base numbers are able to be multiplied (added) without converting to another base at the same base. It means that a number in base k can be multiplied (summed) up with another number in base k without converting into base 10 (for numbers other than base 10) and binary base. A data-oriented memory has been used in this architecture. In this memory, the values relating to numbers and letters have been embedded. As soon as a key is pressed from the keyboard, its ASCII code is compared with the values within the memory in parallel manner, both ASCII code (to take the final result) and binary code are placed in one of the memory cells, if they matched. The input of the multipliers connected to the buses receives their input values from the bus. Binary or decimal multipliers multiply two input values including n levels. When the multiply of two numbers in the first level is greater than the data base, then it subtracts (division) the obtained result from the base and sends one as a carry bit to the second level and this is continued until the last level in the multipliers. The final carry bit is applied to the data-oriented memory with the value of the final result and its ASCII code is placed in the output bus to be shown.

4.1.2 Function unit's architecture

The new Function unit (Shifter + ALU) is similar to the decimal numbers' arithmetic unit, however with this difference that the adder-sub tractor (multiplier-divider), a data-oriented memory in the input of the arithmetic unit and the numbers' base itself are used to division (subtract) and compare in the circuit. The comparator unit is compared with number's base after multiplying two numbers. If the result is greater than or equal to the number's base, then the obtained result is added to the base's two complement (the obtained result is divided with the base) and if it is smaller than the base, then the obtained result is shifted to the output from the zero input of multiplexer without any change. A multiplexer (this multiplexer has been built by using two three-state buffer for speeding up) is used to differentiate the obtained results. This means that if either the carry number or the addition result is greater than the base (the carry number or the addition result is greater than the base of the selection line of multiplexer), the subtraction result is sent to the output because the selection line of the multiplexer becomes one and the input of a multiplexer which is the division (subtraction) result is sent to the output. After obtaining the result, the carry number and the value of the obtained result are applied to the data-oriented memory as a key so that the result is placed in the bus as an ASCII code and is shown in the output equivalent to the ASCII code. If the selection line of the multiplexer becomes zero, means that the multiply of the numbers becomes less than the base, in this case, the obtained result which is in the zero input of the multiplexer is sent to the output.

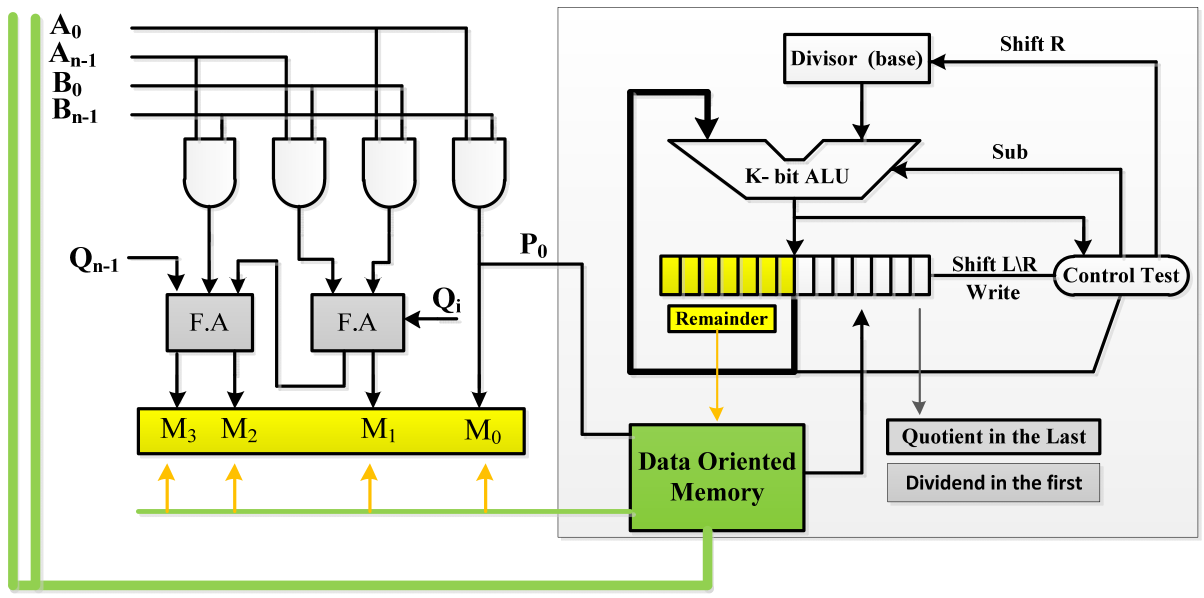

Fig. 10. Architecture of the same base multiplier's arithmetic unit

According to this fact that the above architecture does not have base conversion, but as all adders, it has a delay about two XOR gates for addition and two AND-OR gates for the carry bit in normal case. If the addition result is greater than the base, this delay become two times. In table 1, delay, PDP and power of the adders have been shown with different architectures.

Table 1
Delay, PDP and power

	Power	Delay	PDP
Full adder-1[9]	5.23E-07	7.97E-11	4.17E-17
Full adder-2[10]	4.71E-07	8.82E-11	4.15E-17
Full adder-3[11]	7.12E-07	7.51E-11	5.35E-17

According to the above table, we could use adder architecture with lower power in this design.

Table 2
Dividing methods, Operation, Delay

Dividing methods	Operation	Delay
Comparison	N comparison , Subtraction	2*n+1* = 5
Restored	N Subtraction , Addition	1*n+1* = 3
Nonrestored	Subtraction , Addition	1* +1* = n

In table 2, Complexity and Operation of the dividers have been shown with different architectures. According to the above table, in the new method of the design have used of divider (Non restored).Therefore kinds of divider with the delay was investigated.

5. Conclusion

Today, due to the great importance of time efficiency parameter in problem solving and also due to this fact that memories with so high data storage capability and low cost are available, so we could use data-oriented memories in function unit itself.

The new multiplier's structure introduced in this paper is a type of multiplier- divider which has been designed by adding a data-oriented memory, subtractor and number base for comparison (and one of the inputs of the subtractor (division)) to the multiplier-divider. According to figure11, firstly this design has removed the weakness of the multipliers; secondly it is applicable to the same base numbers. Third, it has led to reduce the number of computations related to base conversion for multiplaction. By reducing the number of computations, the exponent and delay of the arithmetic unit is also reduced and consequently the speed of the system increases (Well as in this design, we were able to reduce number adders to (j-1)k -1 for r bit that in normal mod r = j+k but in new architecture j and k are numbers that could have any bits).

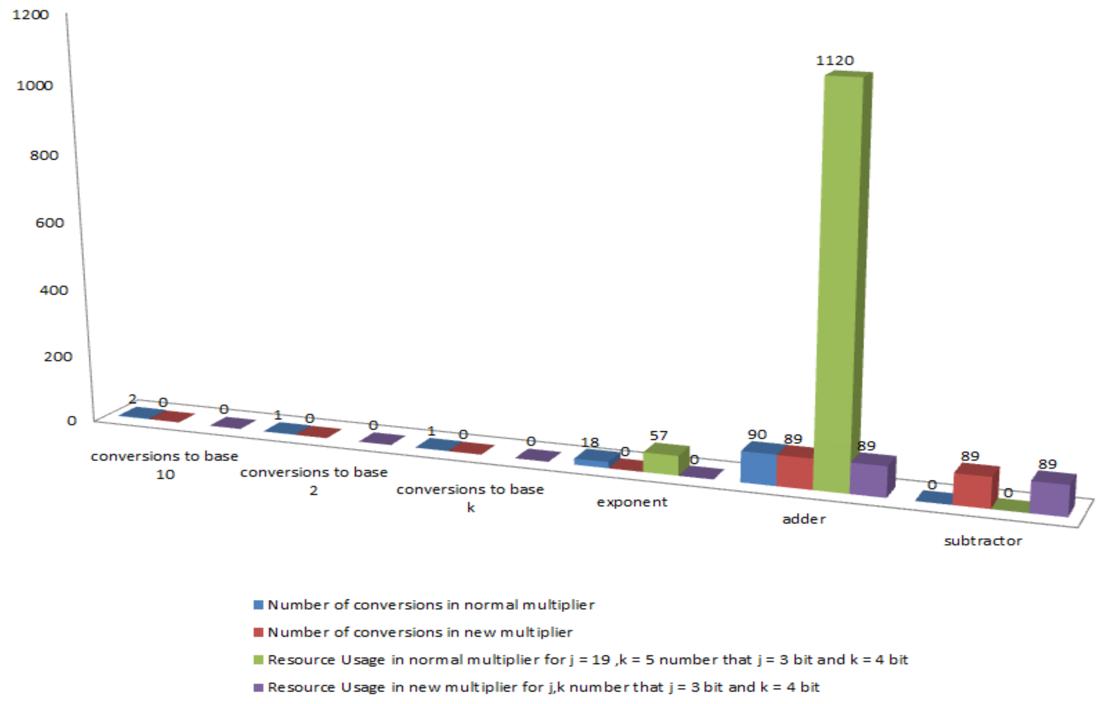

Fig. 11. the amount of conversions and the required time for these conversions.

6. Future works

This multiplier could be use in Embedded Systems, the processing unit of data-oriented memories and also special systems which work with different bases in order to accelerate the computations for example RNS. Also, this multiplier could be applied for speed up exponent operations, logarithmic functions computing to accelerate the computations. Applying this multiplier's algorithm in the software programming libraries, such as Cuda, Java and C++, VHDL, Matlab, and ... leads to reducing the number of programming lines and accelerating the speed of producing output.

References

[1] M. Jafari, et. al (2014). Design of a adderr for similar base numbers without any need for converting base using a data oriented memory. In proceeding of the 21thJulyscientific committee in the frist International Conference on Research in Engineering, Science and Technology.Turkey, Istabnul.

[2] M. Jafari, et. al (2014). Data-oriented memory for storing data irregularly float. In proceeding of the 6thFebruaryEleventh National Conference of Computer and Intelligent Systems. Iran, pp:11\A 5320, Kish.

[3] M. Jafari, et. al (2014). Data-oriented memory for storing data regularly float. In proceeding of the 22thSeptemberTenth National Conference of Computer and Intelligent Systems. Iran, pp:101\A, Kish.

[4] M. Jafari, et. al (2013). Designdata-oriented memory for real values. In proceeding of the 7th october RBPJ2013. Iran, pp:04908, Roudsar.

[5] M. Jafari, et. al (2013). Data-oriented memory for storing data regularly float. In proceeding of the 7th octoberEleventh National Conference of Computer and Intelligent Systems. Iran, pp:11\A 5320, Kish.

[6] M. Morris (2007). Digital Design(Fourth Edition), Tehran, Khorasan.

[7] M. Morris (2010). Computer Architecture(twenty-seventh edition), Tehran, garden.

[8] A.Habibi, et. al (2003). Alternative view of the shortest path problem. In proceeding of the 30th International Mathematical Conference. Iran 1999, pp:122-124. OMG Press, Wily Publishing.

[9] A. Habibi, T. Oladghaffari, M. Mirnia, H. Es-hagi (2010). Data-Oriented Architecture of Sine. International Conforence on Education Technology and Computer.

[10] A. Habibi, T. Oladghaffari, M. Mirnia, Gh. Yaghoubi (2010). Data-Oriented Architecture of Ln function., International Conference on Advanced Computer Control - ICACC.

[11] M. Hosseinzadeh, S. RasouliHeikalabad, A. HabibizadNavin, T. Oladghaffari (2013). MidPoint Memory: A Special Memory Structure For Data Oriented Models Implementation. Journal of Circuits, Systems, and Computers.

[12] M. Hosseinzadeh, S. RasouliHeikalabad, A. HabibizadNavin, T. Oladghaffari (2013). Universal Midpoint Memory: A Special Memory Structure for Data Oriented Models Implementation. Journal of Circuits, Systems, and Computers.

[13] en.wikipedia.org/wiki/Scoreboarding

[14] S. Goel, A. Kumar and M.A. Bayoumi (2006) “Design of robust, energy efficient full adders for deep sub micrometer design using hybrid-CMOS logic style”, IEEE Trans. on VLSI Systems, vol. 14, no. 12, pp.1309-1321.

[15] J. F. Lin, Y. T. Hwang, M. H. Sheu, and C. C. Ho (2007) “A novel high-speed and energy efficient 10-transistor full adder design," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 54, no. 5, pp. 1050- 1059.

[16] Y. Jiang, A. Al-Sheraidah, Y. Wang, E. Sha and J.G. Chung (2004) “A novel multiplexer-based low power

[17] full adder,” IEEE Trans. on Circuits and Systems – II: Express Briefs, vol. 51, no. 7, pp. 345-348.