JOURNAL OF NORTHEASTERN UNIVERSITY

Volume 25 Issue 04, 2022 ISSN: 1005-3026 https://dbdxxb.cn/ Original Research Paper

OPTIMIZATION ANALYSIS OF PPI_MO BASED MATRIX MULTIPLICATION USING IEEE 754 FLOATING POINT MULTIPLIER

[1]Asim Darshan, [2]Ashish Raghuwanshi

Department of Electronics and Communication, IES college of Technology, Bhopal[1,2]

Abstract— In the present scenario, the rapid growth of wireless communication, multimedia applications, robotics and graphics increases the demand for resource efficient, high throughput and low power digital signal processing (DSP) systems. Floating point matrix multiplication is the most widely used fundamental processing element in almost all DSP systems ranging from audio/video signal processing to wireless sensor networks. Hardware implementation of Floating point matrix multiplication requires a huge number of arithmetic operations that affect the speed and consumes more area and power. Pipelining and parallel processing are the two methods used in the DSP systems to reduce the area. Matrix multiplication is the kernel operation used in many transform, image and discrete signal processing application. We develop new algorithms and new techniques for matrix multiplication on configurable devices. In this paper, we have proposed three designs for matrix-matrix multiplication. These design reduced hardware complexity, throughput rate and different input/output data format to match different application needs. The PPI-MO based matrix multiplication is design Xilinx software and simulated number of slice, look up table and delay.

Keywords— IEEE754, Single Precision Floating Point (SP FP), Double Precision Floating Point (DP FP), Matrix Multiplication

1. Introduction

The key mission of this paper is to provide the background material necessary to follow the kernel of this thesis. The wide dynamic range features of Floating point arithmetic system is a common choice for many scientific and signal processing computations [1]. These applications often aim at high performance floating point unit. Basic Linear operations, such as dot product, vector and matrix multiplication are necessary for wide spectrum of computer applications. Floating point matrix multiplication is a building block for many linear algebra kernels [2, 3]. Multiplication is a complex arithmetic operation which is reflected in its relatively high power dissipation, high signal propagation delay, and large area requirement. Hence an efficient multiplier design has become a significant part in VLSI system design. The overall performance of the processing system is determined by the performance of the multiplier. By using an efficient architecture for floating point matrix multiplication improves the computation complexity of the system. In this research work, efficient floating point multipliers are considered for performing matrix multiplication [4].

Floating point multiplication is considered as an abstruse subject though it is found everywhere in a processor. A computer numerical computing capability is characterized with the number of floating point operation per unit time (Mflop/s- Million of floating point operation per second) [5]. Day by day processor is getting more complex in terms of design and its

Submitted: 08/12/2022 **Accepted**: 27/12/2022

performance analysis. In modern electronic systems, a multiplier is a fundamental arithmetic unit and it is extensively used in circuits, for which the multiplication process should be optimized properly. Floating point number system is a common choice for several scientific computations due to its wide dynamic range feature. The word floating point means that, there is no fixed number of digits before and after the decimal point; the decimal point can float. In computation, floating point is the standard representation that approximates a real number so as to support a trade-off between range and precision [6, 7].

Based on IEEE-754 standard, floating point formats are classified into binary and decimal interchange formats. Floating point multipliers are very important in dsp applications. This paper focuses on double precision normalized binary interchange format. Figure 1 shows the IEEE-754 double precision binary format representation. Sign (s) is represented with one bit, exponent (e) and fraction (m or mantissa) are represented with eleven and fifty two bits respectively. For a number is said to be a normalized number, it must consist of'one' in the MSB of the significand and exponent is greater than zero and smaller than 1023. The real number is represented by equations (i) & (2).

$$Z = (-1^{s}) \times 2^{(E-Bias)} \times (1.M) \qquad (1)$$

$$Value = (-1^{signbit}) \times 2^{(Exponent-1023)} \times (1.Mantissa) \qquad (2)$$

Biasing makes the values of exponents within an unsigned range suitable for high speed comparison.

Sign Bit	Biased Exponent	Significand
1-bit	8/11-bit	23/52-bit

Fig. 1: IEEE 754 Single Precision and Double Precision Floating Point Format

IEEE 754 STANDARD FLOATING POINT MULTIPLICATION ALGORITHM

A brief overview of floating point multiplication has been explained below [5-6].

1. Both sign bits S_1, S_2 are need to be Xoring together, then the result will be sign bit of the final product.
2. Both the exponent bits E_1, E_2 are added together, and then subtract bias value from it. So, we get exponent field of the final product.

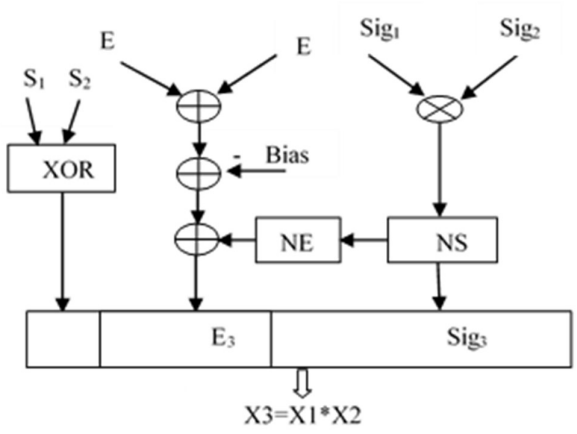

Fig. 2: IEEE754 SP FP and DP FP Multiplier Structure, NE: Normalized exponent, NS: Normalized Significand

II. MATRIX MULTIPLICATION

The basic tool of linear algebra is matrix multiplication. In all computational applications of linear algebra matrix product computation is a central operation. Matrix multiplication is a key computation for several Engineering applications and scientific computing. To the performance of such applications, a fast and efficient implementation of matrix application is critical. Implementation of high performance matrix multiplication can be used to measure the potential performance of the target device [8].

Implementations of high performance level 3 BLAS operations are required in many complex algorithms in digital signal processing, image and video processing applications. Based on the computational applications and performance of the system, many algorithms have been designed for matrix multiplication. There has been wide work for matrix multiplication on parallel algorithms. Two classical algorithms are designed in which, each processor holds consecutive blocks of data based on a square processor grid with a block data distribution. The blocks on one processor are either broad cast to the other processor or transferred to its adjacent processors in the same row throughout the iteration. It developed Parallel universal matrix multiplication algorithm (PUMMA) which provides two-dimensional block cyclic data decomposition for Foxes algorithm. Distribution–Independent Matrix Multiplication Algorithm combines pipeline communication and LCM block concept to achieve the maximum performance. The Scalable Universal Matrix Multiplication Algorithm (SUMMA) for distributed memory concurrent computers. The blocks are broadcast based on flexible broadcast- multiply- roll algorithm; the computation and communication on the processor are overlapped. They parallelized a sequential algorithm on a linear array of processors, which supports massive volume of data transfer on a pipeline optical bus. The proposed rank-1 update algorithm, here sub block of matrices are obtained by multiplying two panels of matrices and handle arbitrary sizes matrices. The Strassen algorithm named after Volker Strassen was used in linear algebra which requires fewer multiplications of matrix elements than the classic matrix multiplication method. It is faster than the standard matrix multiplication algorithms. Strassen

algorithm is a divide and conquer algorithm which partitions the matrix into sub matrices of equal size and employ a divide and conquer strategy on each sub matrices. For a 2×2 matrix multiplication, conventional algorithm requires 8 multiplication operations whereas Strassen's require only 7 multiplication operations and 18 addition or subtractions. The complexity of Strassen algorithm is O(n2) but for classic algorithm the complexity is O(n3). Systolic array is a homogeneous network of tightly coupled data processing units (DPUs) which rhythmically compute a partial result by maintaining a regular flow of data in the network. The DPU process the data received from its upstream neighbours, store the result within itself and passes it down stream. The data stream entering and leaving the DPU are generated by Auto Sequencing Memory Unit (ASM) which include a data counter. DPUs are connected to the neighbour cell in a mesh -like topology and each DPU perform a sequence of processes on the data that flows between them [11, 12].

III. PROPOSED METHODOLOGY

Proposed Parallel-Parallel Input and Multi Output(PPI-MO)

In this design, we opted for faster operating speed by increasing the number of multipliers and registers performing the matrix multiplication operation.

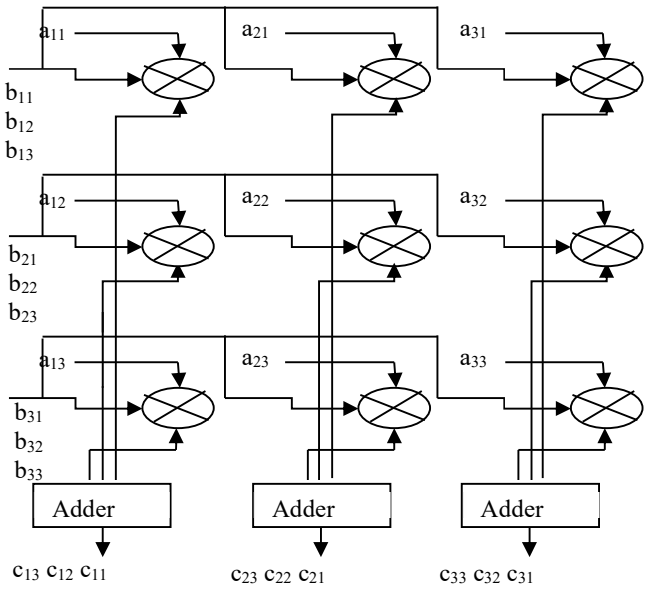

Fig. 3: Proposed PPI – MO Design for n = 3

From equation 2 we have derived for parallel computation of 3 × 3 matrix-matrix multiplication and the structure is shown in figure 3.

For an n×n matrix – matrix multiplication, the operation is performed using n^2 number of multipliers, n^2 number of registers and $n^2 - n$ number of adders. The registers are used to store the partial product results. Each of the n^2 number of multipliers has one input from matrix B and the other input is obtained from a particular element of matrix A.

The dataflow for matrix B is in row major order and is fed simultaneously to the particular row of multipliers such that the i^{th} row of matrix B is simultaneously input to the i^{th} row of multipliers, where 1 < i < n . The elements of matrix are input to the multipliers such that, $(j,i)^{th}$ element of matrix A is input to

The $(i,j)^{th}$ multiplier, where1 < i,j < n. The resultant products from each column of multipliers are then added to give the elements of output matrix C. In one cycle, n elements of matrix C are calculated, so the entire matrix the elements of matrix C are obtained in column major order with n elements multiplication operation requires n cycles to complete.

Let us consider the example of a 3×3 matrix – matrix multiplication operation, for a better analysis of the design (as shown in figure 1). The hardware complexities involved for this design are 9 multipliers, 9 registers and 6 adders. Elements from the first row of matrix B (b_{11} b_{12} b_{13}) are input simultaneously to the first row of multipliers (M_{11} M_{12} M_{13}) in 3 cycles. Similarly, elements from other two rows of matrix B are input to the rest two rows of multipliers. A single element from matrix A is input to each of the multipliers such that, $(j,i)^{th}$ element of matrix A is input to the multiplier M_{ij}, where 1 < i,j < 3. The resultant partial products from each column of multipliers (M_{1k} M_{2k} M_{3k} where 1 < k 3) are added up in the adder to output the elements of matrix C. In each cycle, one column of elements from matrix C is obtained (C_{1k} C_{2k} C_{3k} where1 < k < 3) and so the entire matrix multiplication operation is completed in 3 cycles.

IEEE 754 Floating Point

In IEEE754 standard floating point representation, 8 bit Exponent field in single precision floating point (SP FP) representation and 11 bit in double precision floating point (DP FP) representation are need to add with another 8 bit exponent and 11 bit exponent respectively, in order to multiply floating point numbers represented in IEEE 754 standard as explained earlier. Ragini et al. [10] has used parallel adder for adding exponent bits in floating point multiplication algorithm. We proposed the use of 8-bit modified CSA with dual RCA and 8-bit modified CSA with RCA and BEC for adding the exponent bits. We have found the improved area of 8-bit modified Carry select adder with RCA and BEC over the 8-bit modified CSA with dual RCA.

- Sign bit calculation

To calculate the sign bit of the resultant product for SP FP and DP FP multiplier, the same strategy will work. We just need to XOR together the sign bits of both the operands. If the resultant bit is '1', then the final product will be a negative number. If the resultant bit is '0', then the final product will be a positive number.

- Exponent bit calculation

Add the exponent bits of both the operands together, and then the bias value (127 for SPFP and 1023 for DPFP) is subtracted from the result of addition. This result may not be the exponent

bits of the final product. After the significand multiplication, normalization has to be done for it. According to the normalized value, exponents need to be adjusted. The adjusted exponent will be the exponent bits of the final product.

- Significand bit calculation

Significand bits including the one hidden bit are need to be multiply, but the problem is the length of the operands. Number of bits of the operand will become 24 bits in case of SP FP representation and it will be 53 bits in case of DP FP representation, which will result the 48 bits and 106 bits product value respectively. In this paper we use the technique of break up the operands into different groups then multiply them. We get many product terms, add them together carefully by shifting them according to which part of one operand is multiplied by which part of the other operand. We have decomposed the significand bits of both the operands ain four groups. Multiply each group of one operand by each group of second operand. We get 16 product terms. Then we add all of them together very carefully by shifting the term to the left according to which groups of the operands are involved in the product term.

IV. Simulation Result

All the designing and experiment regarding algorithm that we have mentioned in this paper is being developed on Xilinx 14.2i updated version. Xilinx 6.2i has couple of the striking features such as low memory requirement, fast debugging, and low cost. The latest release of ISE™ (Integrated Software Environment) design tool provides the low memory requirement approximate 27 percentage low. ISE 14.2i that provides advanced tools like smart compile technology with better usage of their computing hardware provides faster timing closure and higher quality of results for a better time to designing solution.

For parallel in multiple out shift registers, all data bits appear on the parallel input immediately following the simultaneous entry of the date bits. Four-bit parallel in multiple out shift register is constructed by four D flip-flops.

In fig. 4 and fig. 5 have shown the resistor transistor logic (RTL) using 3×3 PPI-MO matrix multiplication and output waveform of 3×3 PPI-MO matrix multiplication respectively.

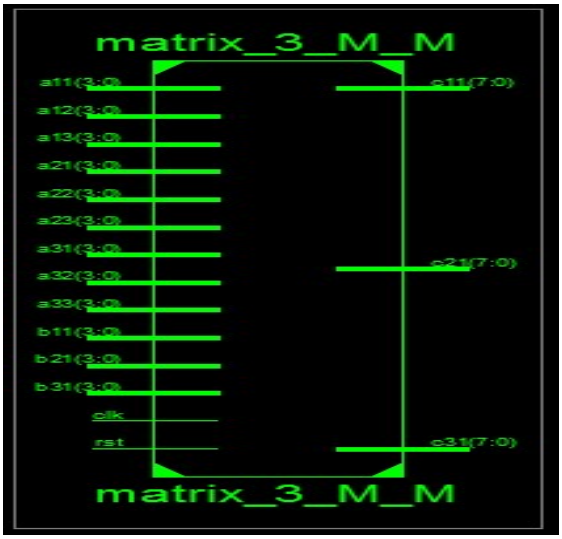

Fig. 4: View Technology Schematic of 3×3 Matrix Multiplications using PPI-MO

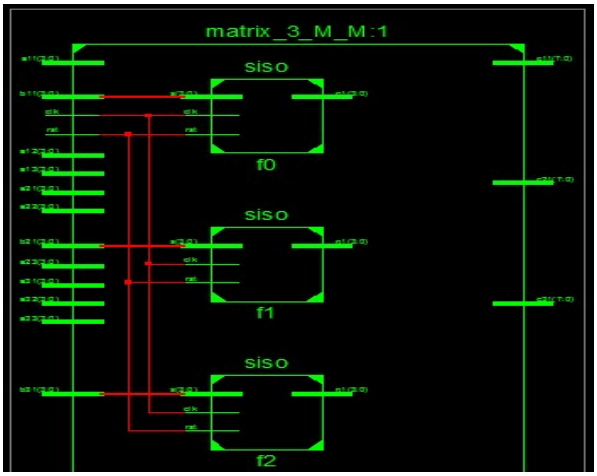

Fig. 5: View Technology Schematic of 3×3 Matrix Multiplications using PPI-MO

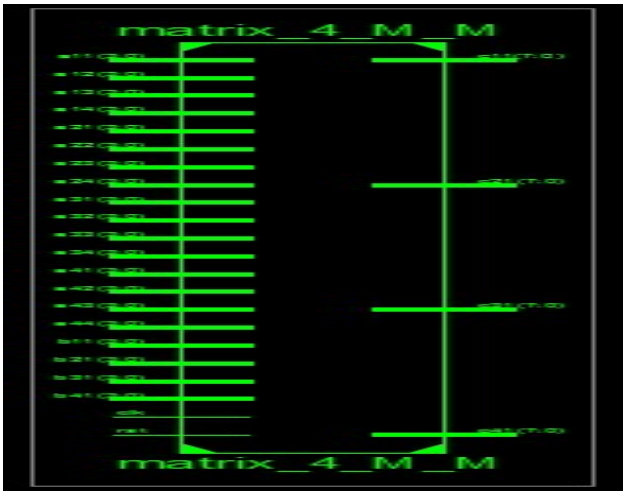

Fig. 6: View Technology Schematic of 4×4 Matrix Multiplications using PPI-MO

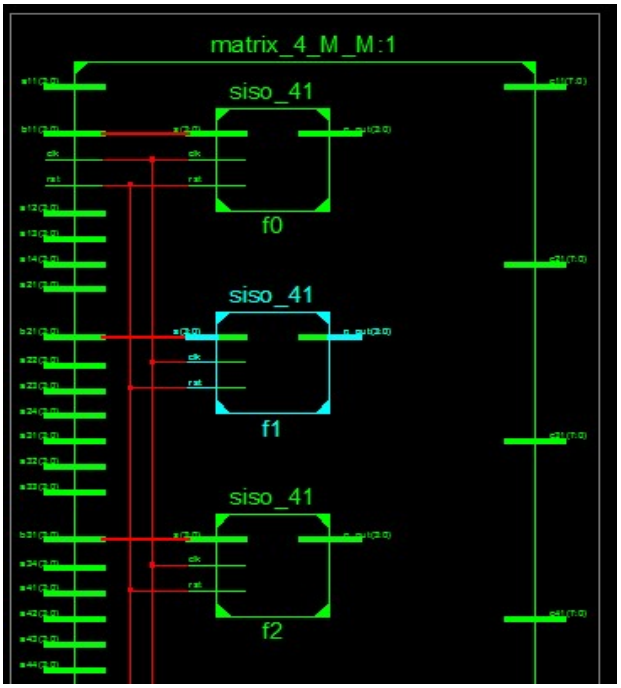

Fig. 7: View Technology Schematic of 4×4 Matrix Multiplications using PPI-MO

Table I: Comparison Result

Parameter	Previous SPFP Algorithm	Implemented SPFP Multiplier using Partition Method	Previous DPFP Algorithm	Implemented DPFP Multiplier using Partition Method
Number of Slice LUTs	705	226	5153	682
Number of Input Output Bonded	96	96	192	192
Maximum Combinational Path Delay	44.823 ns	33.97 ns	83.169 ns	70.70 ns

Table 2: Simulation result for 3×3 and 4×4 Matrix Multiplication

Structure	Dimension	Slice	LUTs	IOBs	Delay (ns)
MM using PPI-SO	3×3	44	15	34	11.222
MM using PPI-MO		93	154	74	15.058
MM using PFI-MO		34	55	38	9.128
Previous Design [1]	4×4	311	212	-	16.513
MM using PPI-SO		49	88	42	13.771
MM using PPI-MO		221	388	74	15.058
MM using PFI-MO		39	72	48	11.543

V. Conclusion

The floating point matrix multiplication for rank-1 update algorithm is designed using Urdhva Tiryagbhyam multiplier based on pipeline processing. This proposed arithmetic level reduction technique achieves area efficient matrix multiplication with reduced delay. The efficiency of

the proposed method is experimented using Xilinx 14.2i simulating tool. The block matrix multiplication architecture is designed using Strassen matrix multiplication algorithm and Urdhva Tiryagbhyam multiplier based on parallel processing. The Strassen algorithm using divide and conquer method divides the matrix into 4×4 matrix and applies PPI_MO which follows vertical and cross multiplications. By proper scheduling and reuse of available resources the proposed algorithm increases the performance.

REFRENCES

[1] Chen Yang;Siwei Xiang;Jiaxing Wang;Liyan Liang, "A High Performance and Full Utilization Hardware Implementation of Floating Point Arithmetic Units", 28th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), IEEE 2021.

[2] Rongyu Ding;Yi Guo;Heming Sun;Shinji Kimura, "Energy-Efficient Approximate Floating-Point Multiplier Based on Radix-8 Booth Encoding", IEEE 14th International Conference on ASIC (ASICON), IEEE 2021.

[3] Wei Mao;Kai Li;Xinang Xie;Shirui Zhao;He Li;Hao Yu, "A Reconfigurable Multiple-Precision Floating-Point Dot Product Unit for High-Performance Computing", Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE 2021.

[4] Rahul Rathod;P Ramesh;Pratik S Zele;Annapurna K Y, "Implementation of 32-Bit Complex Floating Point Multiplier Using Vedic Multiplier, Array Multiplier and Combined integer and floating point Multiplier (CIFM)", International Conference for Innovation in Technology (INOCON), IEEE 2020.

[5] S. Ross Thompson;James E. Stine, "A Novel Rounding Algorithm for a High Performance IEEE 754 Double-Precision Floating-Point Multiplier", 38th International Conference on Computer Design (ICCD), IEEE 2020.

[6] P.L. Lahari;M. Bharathi;Yasha Jyothi M Shirur, "High Speed Floating Point Multiply Accumulate Unit using Offset Binary Coding", 7th International Conference on Smart Structures and Systems (ICSSS), IEEE 2020.

[7] Lakshmi kiran Mukkara and K.Venkata Ramanaiah, "A Simple Novel Floating Point Matrix Multiplier VLSI Architecture for Digital Image Compression Applications", 2nd International Conference on Inventive Communication and Computational Technologies (ICICCT 2018) IEEE.

[8] Soumya Havaldar, K S Gurumurthy, "Design of Vedic IEEE 754 Floating Point Multiplier", IEEE International Conference On Recent Trends In Electronics Information Communication Technology, May 20-21, 2016, India.

[9] Ragini Parte and Jitendra Jain, "Analysis of Effects of using Exponent Adders in IEEE-754 Multiplier by VHDL", 2015 International Conference on Circuit, Power and Computing Technologies [ICCPCT] 978-1-4799-7074-2/15/$31.00 ©2015 IEEE.

[10] Ross Thompson and James E. Stine, "An IEEE 754 Double-Precision Floating-Point Multiplier for Denormalized and Normalized Floating-Point Numbers", International conference on IEEE 2015.

[11] M. K. Jaiswal and R. C. C. Cheung, “High Performance FPGA Implementation of Double Precision Floating Point Adder/Subtractor”, in International Journal of Hybrid Information Technology, vol. 4, no. 4, (2011) October.
[12] B. Fagin and C. Renard, "Field Programmable Gate Arrays and Floating Point Arithmetic," IEEE Transactions on VLS1, vol. 2, no. 3, pp. 365-367, 1994.
[13] N. Shirazi, A. Walters, and P. Athanas, "Quantitative Analysis of Floating Point Arithmetic on FPGA Based Custom Computing Machines," Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM"95), pp.155-162, 1995.
[14] Malik and S. -B. Ko, “A Study on the Floating-Point Adder in FPGAs”, in Canadian Conference on Electrical and Computer Engineering (CCECE-06), (2006) May, pp. 86–89.
[15] D. Sangwan and M. K. Yadav, “Design and Implementation of Adder/Subtractor and Multiplication Units for Floating-Point Arithmetic”, in International Journal of Electronics Engineering, (2010), pp. 197-203.
[16] L. Louca, T. A. Cook and W. H. Johnson, “Implementation of IEEE Single Precision Floating Point Addition and Multiplication on FPGAs”, Proceedings of 83rd IEEE Symposium on FPGAs for Custom Computing Machines (FCCM‟96), (1996), pp. 107–116.
[17] Jaenicke and W. Luk, "Parameterized Floating-Point Arithmetic on FPGAs", Proc. of IEEE ICASSP, vol. 2, (2001), pp. 897-900.
[18] Lee and N. Burgess, “Parameterisable Floating-point Operations on FPGA”, Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems, and Computers, (2002).
[19] M. Al-Ashrafy, A. Salem, W. Anis, “An Efficient Implementation of Floating Point Multiplier”, Saudi International Electronics, Communications and Photonics Conference (SIECPC), (2011) April 24-26, pp. 1-5.