VLSI Architecture for Matrix Multiplier using Parallel To Parallel Input Multiple Output Technique

Rajesh Yadav[1], Prof. Satyarth Tiwari[2]
M. Tech. Scholar, Department of Electronics and Communication, Bhabha Engineering Research Institute, Bhopal[1]
Guide, Department of Electronics and Communication, Bhabha Engineering Research Institute, Bhopal[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. Matrix multiplication (MM) 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 MM 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. MM is the kernel operation used in many transform, image and discrete signal processing application. We develop new algorithms and new techniques for MM on configurable devices. In this paper, we have proposed MM using round based approximated multipliers. This design reduced hardware complexity, delay and input/output data format to match different application needs. The PPI-MO based MM is design Xilinx software and simulated number of slice, look up table and delay.

Keywords—Matrix Multiplication, Parallel to Parallel Input Multiple Output (PPI-MO), Round based Approximated Multipliers (ROAM)

1. Introduction

For engineering applications and various scientific computing, matrix multiplication is a fundamental computation. To improve the performance of such applications, a fast and efficient matrix multiplication algorithm is required. This can be accomplished by designing an efficient multiplier with parallel and pipelined architectures [1]. In parallel processing the performance can be increased by executing many floating point operations simultaneously or in parallel. The parallelization strategy allows the use of many processing elements in parallel. Pipelining is a technique in which multiple floating point operations are overlapped in execution. Scheduling process reorders the execution order of floating point operation so as to avoid data hazard. Reliable and area efficient Urdhva Tiryagbhyam and Strassen multiplier architectures are designed to improve the performances of the floating point unit [2, 3]. The major problems faced by most of the multipliers are delay and area. By providing the proper pipelining process and reuse of the available components, the overall performances of the system can be improved. The multiplier unit for large number performance is improved by using Karatsuba and Urdhva Tiryagbhyam algorithm combination. Hence the effective methods are found to design an architecture that improves the performance by complete utilization of the available resources [4].

The complexity of matrix multiplication has attracted a lot of attention in the last forty years. In this paper we will consider matrix multiplication as the problem, give various methods to solve this problem and find the best one that takes the least time.

Matrix multiplication is the kernel of many scientific applications [5, 6]. It is a binary operation that takes a pair of matrices, and produces another matrix. If A is an n-bym matrix and B is an m-by-p matrix, the result AB of their multiplication is an n-by-p matrix defined only if the number of columns m of the left matrix A is the equal to the number of rows of the right matrix B. The result of matrix multiplication is a matrix whose elements are found by multiplying the elements within a row from the first matrix by the associated elements within a column from the second matrix and summing the products [7]. The procedure for finding an element of the resultant matrix is to multiply the first element of a given row from the first matrix times the first element of a given column from the second matrix, then add to that the product of the second element of the same row from the first matrix and the second element of the same column from the second matrix, then add the product of the third elements and so on, until the last element of that row from the first matrix is multiplied by the last element of that column from the second matrix and added to the sum of the other products [8].

Ex:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

As we mentioned before there are many methods to calculate the multiplication of matrixes. All of them give the same result but each one consumes different space in memory and takes different processor time. The methods that we will test are:

1. Row by Column method
2. Row by Row method
3. Column by Column method
4. Strassen method

II. 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.

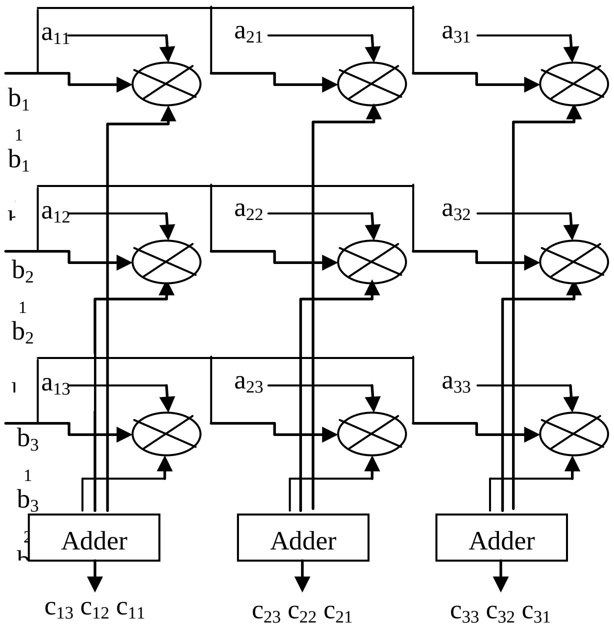


Fig. 1: Proposed PPI – MO Design for n = 3

We have derived for parallel computation of 3 × 3 matrix-matrix multiplication and the structure is shown in figure 1.

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.

Rounded Based Approximated Multipliers

ROBA is incredibly desirable to achieve this minimization with the least amount of output (speed) penalty possible. These portable devices' digital image processing (DSP) blocks are essential for understanding a variety of multimedia applications. The arithmetic logic unit is the mathematical centre of these blocks, including multiplications responsible for the number of arithmetic operations in these DSP systems. RoBA multiplier offers a certified, unused, high output, high speed and energy effective rounding multiplier [9]. We give a fast speed, energy-efficient estimate multiplier. The theoretical procedure refers to multiplications, both signed and non-signed. The estimated multiplier requires three hardware design units, one non-signed and two signed. By contrasting their output with that of some estimated and reliable multipliers using various design criteria, the efficacy of the suggested multipliers is evaluated. Also, two framework images processing (sharping and smoothing) are researching the utility of the estimated multiplier suggested [10].

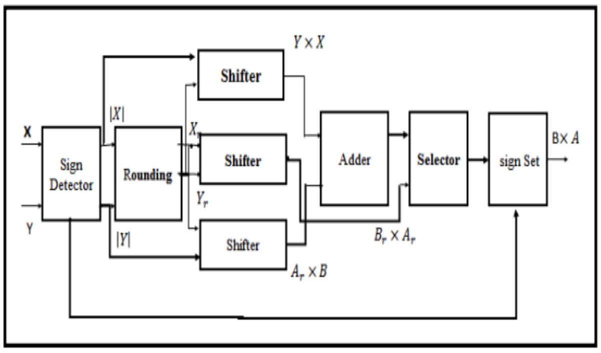


Fig. 2: Block diagram of ROBA Multiplier

III. SIMULATION RESULT

A FPGA (Field Programmable Gate Array) is an incorporated circuit comprising of an assortment of rationale squares, I/O cells and interconnection assets and this permits the chip to be reconfigured to associate the sources of info and yields (I/O) and rationale squares together from various perspectives. The representations of the permanent points and floating points are generally used in numerous applications. It is mainly applicable for designing process of the DSP applications. Also, floating point is demonstrated as very small to large numbers, which employed with the improved range. Every rationale square has customarily the capacity to do a basic rationale activity, for example, AND or XOR, and for the most part contains some level of memory, it be a straightforward flip-flop or a progressively intricate square of memory. The rationale squares have developed to be more rationale work squares utilizing query tables inside the squares to switch the current capacity; to perform assignments such math tasks.

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. 3 and fig. 4 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.

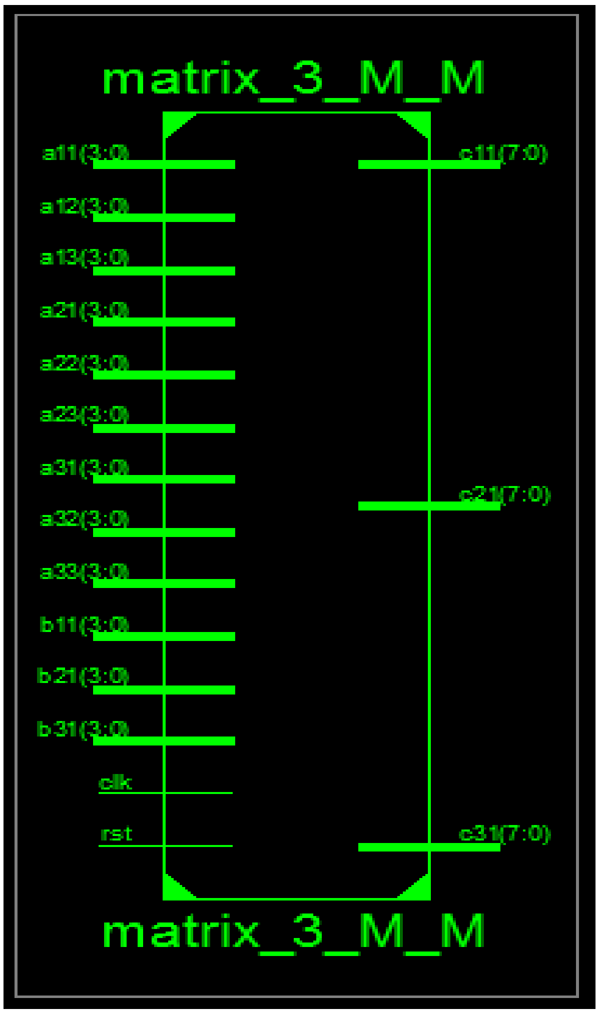


Fig. 3: View Technology Schematic of 3×3 Matrix Multiplications using PPI-MO

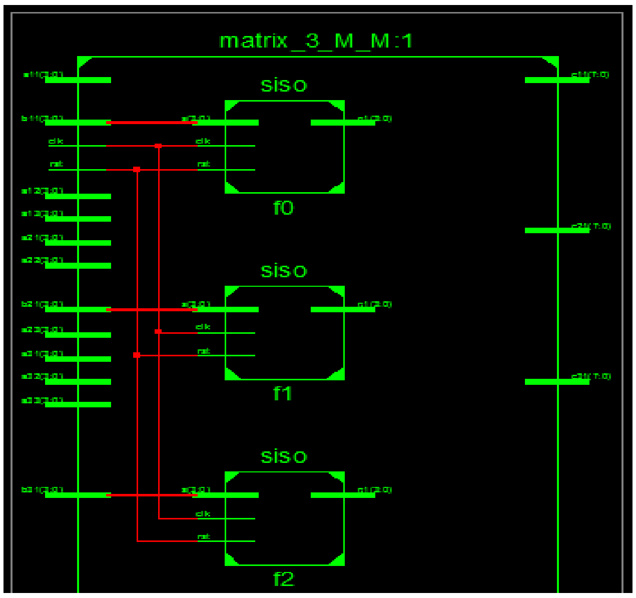
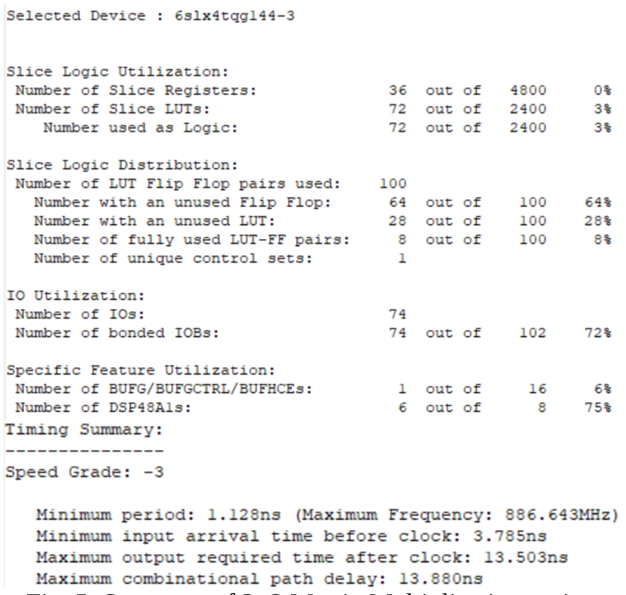


Fig. 4: View Technology Schematic of 3×3 Matrix Multiplications using PPI-MO


```
Selected Device : 6slx4tqg144-3


Slice Logic Utilization:
 Number of Slice Registers:              36  out of   4800     0%
 Number of Slice LUTs:                   72  out of   2400     3%
    Number used as Logic:                72  out of   2400     3%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    100
   Number with an unused Flip Flop:      64  out of    100    64%
   Number with an unused LUT:            28  out of    100    28%
   Number of fully used LUT-FF pairs:     8  out of    100     8%
   Number of unique control sets:         1

IO Utilization:
 Number of IOs:                          74
 Number of bonded IOBs:                  74  out of    102    72%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRL/BUFHCEs:         1  out of     16     6%
 Number of DSP48A1s:                      6  out of      8    75%
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 1.128ns (Maximum Frequency: 886.643MHz)
   Minimum input arrival time before clock: 3.785ns
   Maximum output required time after clock: 13.503ns
   Maximum combinational path delay: 13.880ns
```
Fig. 5: Summary of 3×3 Matrix Multiplications using PPI-MO

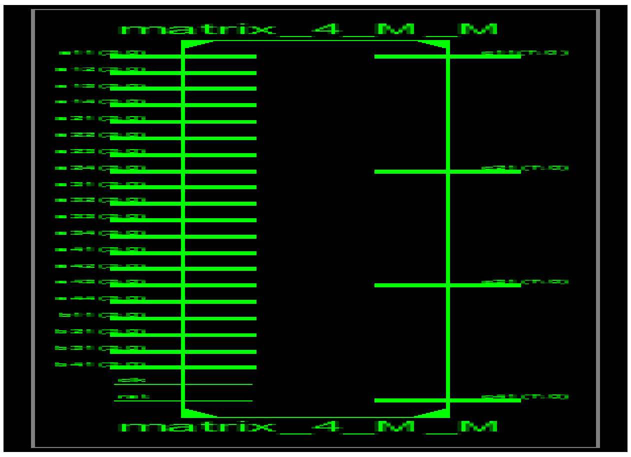


Figure 6: View Technology Schematic of 4×4 Matrix Multiplications using PPI-MO

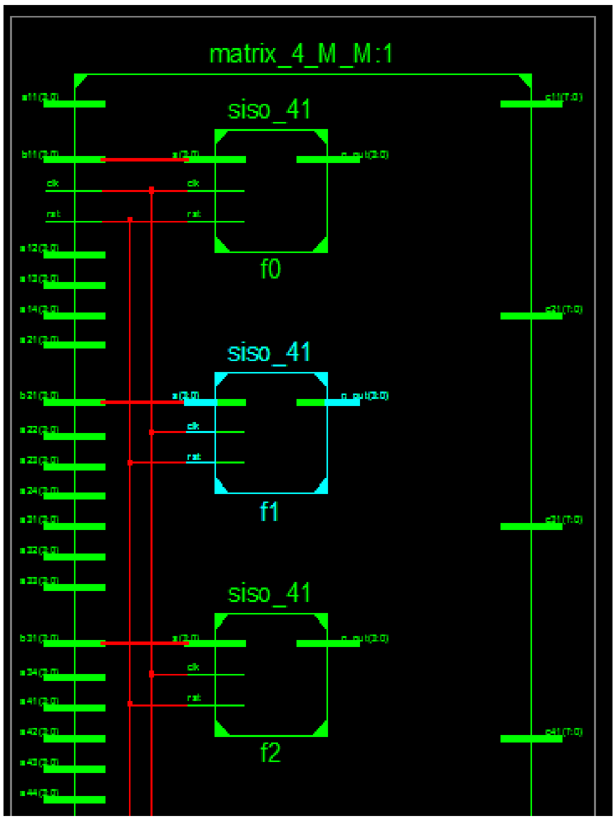
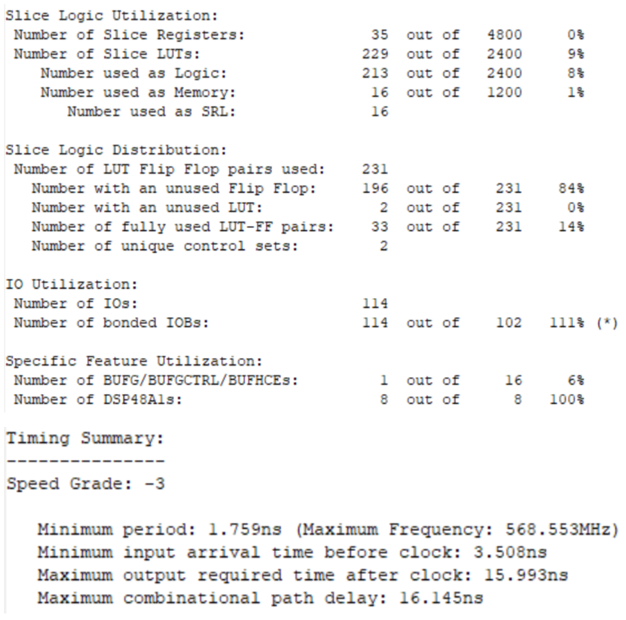


Fig. 7: View Technology Schematic of 4×4 Matrix Multiplications using PPI-MO



```
Slice Logic Utilization:
 Number of Slice Registers:                35  out of   4800    0%
 Number of Slice LUTs:                    229  out of   2400    9%
    Number used as Logic:                 213  out of   2400    8%
    Number used as Memory:                 16  out of   1200    1%
       Number used as SRL:                 16

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:      231
   Number with an unused Flip Flop:       196  out of    231   84%
   Number with an unused LUT:               2  out of    231    0%
   Number of fully used LUT-FF pairs:      33  out of    231   14%
   Number of unique control sets:           2

IO Utilization:
 Number of IOs:                           114
 Number of bonded IOBs:                   114  out of    102  111% (*)

Specific Feature Utilization:
 Number of BUFG/BUFGCTRL/BUFHCEs:           1  out of     16    6%
 Number of DSP48A1s:                        8  out of      8  100%

Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 1.759ns (Maximum Frequency: 568.553MHz)
   Minimum input arrival time before clock: 3.508ns
   Maximum output required time after clock: 15.993ns
   Maximum combinational path delay: 16.145ns
```

Fig. 8: Summary of 3×3 Matrix Multiplications using PPI-MO

IV. CONCLUSION

From the design analysis, it is inferred that the parallel matrix multiplication with ROAM multipliers consumes less area and delay compared to previous algorithm which is designed using array multiplier based on pipeline processing The present investigation is based on the area, delay and power consumption with promising results. To reduce the row on a matrix, a series of row processes are performed to transform the matrix until the lower left hand end of the matrix is occupied with zeros. Three basic row operations performed are swap any two rows of the input matrices and multiply a non-zero constant to a row and add scalar multiple of one row to another.

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.