http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

Information Technology Journal 12 (9): 1832-1838, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.1832.1838
© 2013 Asian Network for Scientific Information

An Optimized Floating-Point Matrix Multiplication on FPGA

[1]Ting Zhang, [1]Cheng Xu, [1]Tao Li, [2]Yunchuan Qin and [1]Min Nie
[1]Embedded System and Network Laboratory, College of Information Science and Engineering, Hunan University, Changsha, 410082, China
[2]College of Information Science and Engineering, Hunan University, Changsha, 410082, China

Abstract: Matrix multiplication is a kernel and fundamental operation in many applications including image, robotic and digital signal processing. The key component of matrix multiplication is Multiplier Accumulator (MAC) which is a decisive component for the performance of matrix multiplication. This study proposes a pipelined floating-point MAC architecture on Field Programmable Gate Array (FPGA) using a novel accumulating method. By adding the last N-stage results of the pipelined adder, the accumulation of the multiplier products can be obtained. Then, a matrix multiplication is implemented by employing parallel systolic structure based on the proposed MAC. Experimental results demonstrate that the proposed MAC architecture achieves higher clock speed and consumes less hardware resources than previous designs and the matrix multiplier has a good performance and scalability. It also can be concluded that the efficiency of the matrix multiplier is even higher when the matrices are larger.

Key words: Matrix multiplication, multiplier accumulator, floating-point, pipeline

INTRODUCTION

Matrix multiplication is a complex and fundamental matrix operation in many algorithms used in scientific computations. It is a frequently used kernel operation in a wide variety of computer vision, robotics and digital signal processing (Beauchamp *et al*., 2008; Qasim *et al*., 2008; Yang *et al*., 2012). Thus, efficient matrix multiplier plays a significant role in improving the performance of these applications. However, some applications have a rigorous demand on real-time and the matrix elements of them are floating-point (Luo and Martonosi, 2000; Zhu, 2013). Traditionally, matrix multiplication operation is implemented on PC or DSP (Digital Signal Processor) which is based on serial structure (Riaz *et al*., 2005; Xu *et al*., 2011) and often become the bottleneck of the overall system. Due to the programmability, density increasing and massive computing performance especially in floating-point calculation, Field Programmable Gate Array (FPGA) is becoming a promising way to speed-up the floating-point matrix multiplication.

At present, researches on FPGA based matrix multiplication have made some achievements. A large number of previous works have been done to the design and implementation of fixed-point matrix multiplication. Amira *et al.* (2000) implemented an 8 bit fixed-point matrix multiplication but the bandwidth proportionally increased with the size of matrices which limits its scalability by hardware resources. Jang *et al*. (2005) presented an algorithm which needs fixed bandwidth but the memory size is proportional to the matrices size. As in the field of floating-point matrix multiplication, Zhuo and Prasanna (2007) presented an algorithm which employs linear systolic architecture to implement the calculation but it requires intercommunication between the neighboring PEs (Processing Element), such communication limits the algorithm's extensibility. Campbell and Khatri (2006) proposed a parallel systolic structure which avoids the intercommunication, this structure can deal with multiple PE simultaneously and improve computing efficiency, (Dou *et al*., 2005; Paidimarri *et al*., 2009; Fovanovic and Milutinovic, 2012) adopted the structure but the efficiency of their design is not ideal.

The key component of the matrix multiplication is Multiplier Accumulator (MAC) which consists of multipliers and adders (Jin *et al*., 2006). Some researchers have attempted to improve the performance of MAC by optimizing the architecture of the multiplier and adder but the improvement is not enough. Some others use IP (Intellectual Property) core of the multiplier and adder to realize the MAC. The IP core has been optimized according to the FPGA chip technology which lays a good foundation for designing a High-performance MAC. With the optimized IP core of floating-point multiplier and

Corresponding Author: Ting Zhang, College of Information Science and Engineering, Hunan University, Changsha, Hunan, 410082, China Tel: 86-135-48606167

adder, a signal clock cycle method was adopted in (Tian *et al*., 2008) but the speed of this design is low since it is not a pipelined structure. Liu *et al*. (2012) proposed a loop pipeline structure which is used to solve the problem of data sequence conflict in their design but such extra control process consumes a large number of registers. All the methods mentioned above don't take full advantages of the optimized IP core according to the computing process of MAC.

This study proposes a novel architecture for pipelined floating-point MAC by splitting the multiply accumulation process. Then with the MAC, a large scale configurable floating-point matrix multiplier adopting the parallel systolic architecture is implemented which can bring a significant improvement to the computing performance of matrix multiplication.

Matrix multiplication: In general, assuming the dimension of matrix A and matrix B are $m\times l$ and $l\times n$, matrix multiplication $C = A\times B$ can be defined as following:

$$c_{ij}=\sum_{k=1}^{l} a_{ik}\times b_{kj}; 1\le i\le m, 1\le j\le n \tag{1}$$

where, matrix $A = (a_{ik})$, $B = (b_{kj})$ and $C = (c_{ij})$. The column size of matrix A must equal to the row size of matrix B and the dimension of matrix C is $m\times n$. The matrix C can be calculated as shown in Fig. 1.

In the procedure, the calculation for every element of matrix C consists of $l\times l$ multiplications and $l\times l$ additions, so the whole process includes $2\times m\times n\times l$ multiplications and additions and the computational complexity is $O(n^3)$. The calculation of floating-point matrix multiplication will be more complex, as the floating-point multiplier and adder need several operations to process exponent and mantissa. This matrix multiplication procedure is time consuming if it is executed in the mentioned serial structure. So it is necessary to design a parallel architecture to improve the performance of the matrix multiplication.

Architecture and design: In matrix multiplication, MAC is the most important component. The performance of MAC determines the efficiency of the matrix multiplication directly. A matrix multiplier usually contains several MAC and each MAC is called a PE. The study proposes a novel architecture for pipelined floating-point MAC and implements matrix multiplier based on the MAC in parallel architecture.

The MAC architecture: MAC is composed of one floating-point multiplier and one accumulator, the

```
Function MatrixMutiplication (Matrix A, Matrix B)
/*Matrix A: m×l-dimension matrix*/
/*Matrix B: l×n-dimension matrix*/
/*Matrix C: m×n-dimension matrix*/
{
  Matrix C;
  For (int i = 0; i<m; I++)
        For (int j = 0; j<n; j++)
                $C_{ij}\leftarrow 0$
  For (int i = 0; i<m; I++)
        For (int j = 0; j<n; j++)
              For (int k = 0; k<l; k++)
                    $C_{ij}\leftarrow C_{ij}+A_{ik}\times B_{kj}$
}
```

Fig. 1: Pseudo code of computing process in matrix multiplication

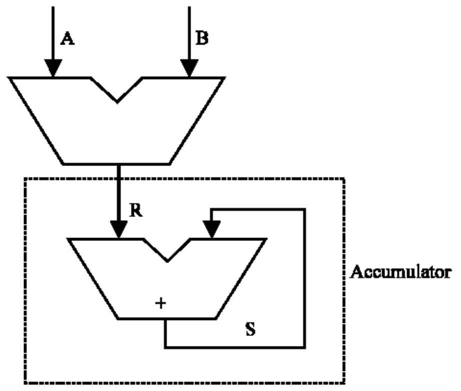

Fig. 2: Frame structure of MAC, MAC: Multiplier Accumulator, A and B: Input of the multiplier, R: Result of the multiplier and S: Result of the adder

accumulator includes only one floating-point adder. Figure 2 shows the structure of the MAC, in which, A and B are inputs of the multiplier, R is the result of the multiplier which is also the input of the accumulator, C is the result of the adder. During each clock cycle, the accumulator continuously takes C and R as the inputs of the adder to complete the accumulation operation.

The optimized IP core of floating-point multiplier and adder are all pipelined structure, so the MAC designed with them can continuously receive the input data and export the result sequentially. Supposing the pipeline stage of the floating-point multiplier and adder are M and N, respectively, the computing result of the input in the first clock cycle will arrive at clock cycle M+N+1. Then the second clock cycle will arrive at clock cycle M+N+2. So it is easy to find that the computing result of the input in the clock cycle P will arrive at clock cycle M+N+P.

Figure 3 illustrates the computing process of the accumulation in MAC. As shown in the figure, N is the

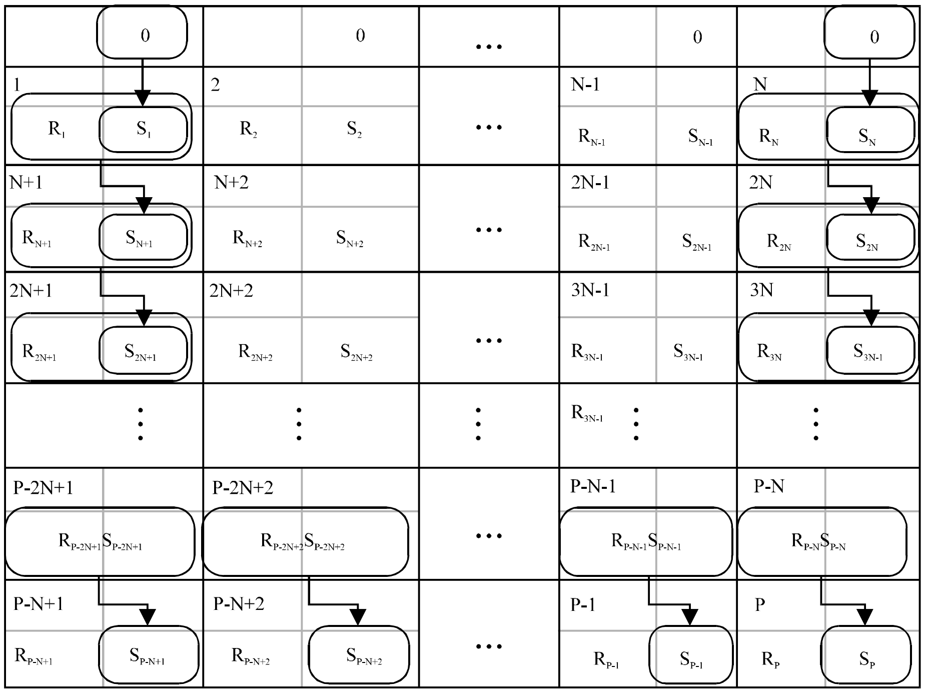

Fig. 3: Accumulation process of the pipeline accumulator in MAC, MAC: Multiplier Accumulator, Ri: Result of multiplier in clock cycle i, Si: Result of adder in clock cycle i and N: Pipeline stages of the adder

pipeline stage of the adder, the top left of each grid is clock cycle, R and S are the input and the output of accumulator in current clock cycles. The green arrows represent the direction of data flow and the red rounded rectangles are the operator of a multiply addition operation. The initial value of the accumulation is zero, so the results of the accumulator are all zero from clock cycle 1 to N which can be expressed as:

$$S(i) = 0; (1 \le i \le N) \qquad (2)$$

From clock cycle N+1, the accumulator begins to export valid result. And at the clock cycle N+1, the result of the accumulator can be written as:

$$S(N+1) = R(1)+S(1) \qquad (3)$$

In the equation above, as S (1) = 0, it can be obtained as:

$$S(N+1) = R(1) \qquad (4)$$

At the clock cycle 2N+1, the result of the accumulator can be written as:

$$S(2N+1) = S(N+1)+R(N+1) = R(1)+R(N+1) \qquad (5)$$

Then the result of the accumulator at clock cycle i (i>N) can be expressed by the following equation:

$$S(i) = S(i-N)+R(i-N) = \sum_{j=1}^{M} R(i-j\times N); (1 \le M \le \lfloor (i/N) \rfloor) \qquad (6)$$

According to the analysis above, the following equation can be figured out:

$$Q = S(P)+S(P-1)+S(P-2)+\cdots+S(P-N) = \sum_{j=1}^{P} R(j) \qquad (7)$$

As shown in the equation, Q is the summations of all the multiplication results.

To obtain the final result of the accumulation, one has to add the results of the last N stages. As using multiple adders to implement the addition is hardware resources consuming and has no obvious improvement on speed, the proposed architecture uses a pipelined structure with only one floating-point adder to complete the addition of the last N data instead of multiple adders.

Parallel architecture of matrix multiplier: In order to achieve a parallel matrix multiplier, the data should be read in parallel according to the computing processes of matrix multiplication. As the row size of matrix A equals to column size of matrix B, the data from them can be read simultaneously to perform the multiply accumulation operation and calculate the corresponding elements of matrix C. If the number of the column of matrix A is *l*, Fig. 4 shows the sequence of the data read from matrices A and B from clock cycle 1 to *l*, then these data will be fed into MAC to in this order.

To achieve the reading and computing in parallel, each row of matrix A and each column of matrix B in memory A array and memory B array should be stored, respectively. Figure 5 presents the parallel array architecture of matrix multiplication. In this architecture, RAM A_i stores the i-th row of matrix A, RAM B_j stores the jth column of matrix B. Each PE receives elements read from RAM A and RAM B simultaneously and calculates each element of matrix C. Thus for a m×n matrix multiplier, m×n multiplier accumulation operations are performed in each clock cycle simultaneously, the computational complexity can be reduced to O(n).

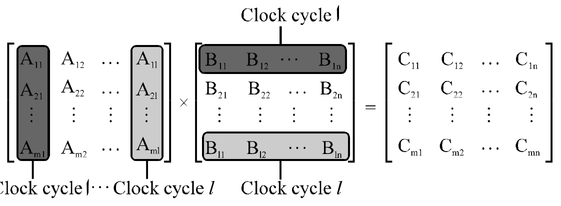

Fig. 4: Reading process of data sequence from clock cycle1 to *l*

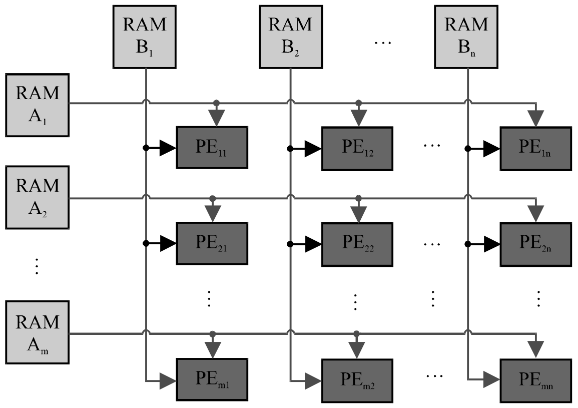

Fig. 5: Parallel array architecture of matrix multiplier, PE: Processing element

Since, there is no intercommunication between the neighboring PEs, they complete their own computation task independently, so the architecture can be applied to matrix multiplication with any size. It also can easily be extended to more pieces of FPGA parallel computing.

Experiment and evaluation: The proposed MAC architecture and matrix multiplier is implemented by Hardware Description Language (HDL). These designs are simulated with Modelsim 6.5 b and synthesized with Quartus II 11.1 on an Altera Stratix III based evaluation board. Several evaluations are made, respectively, to verify the performances of the proposed MAC and matrix multiplier.

Performance of MAC: As pipeline technology decomposes a single long operation into multiple shorter independent stages, it can improve the execution frequency of the overall system but at the cost of extra resources. So pipeline stage of the floating-point multiplier and adder is varied in the experiment to evaluate the performance and resources consumption of the proposed MAC. So far, several related works have been published (Paidimarri *et al.*, 2009; Jin *et al.*, 2006; Liu *et al.*, 2012). Paidimarri *et al.* (2009) designed a pipelined MAC with single-cycle accumulation and it was synthesized on the same FPGA device with this study. Table 1 shows the comparison with (Paidimarri *et al.*, 2009) in frequency and resources consumption under different pipeline stages. It is observed that the frequency of the proposed MAC and (Paidimarri *et al.*, 2009) increases with the increasing of the number of pipeline stages, meantime the resources consumption also increases. The proposed MAC can achieve a frequency at 262.95 MHZ and save about 17.5% Look-Up-Table (LUT) resources when the pipeline stage is 9. When the pipeline stage is 11, the frequency is 265.67 MHZ and the LUT resources can be saved about 49.4%. Compared with (Paidimarri *et al.*, 2009), the proposed MAC can achieve higher frequency while utilizing less resources consumption.

Table 2 presents the MAC performances comparison with (Jin *et al.*, 2006; Dou *et al.*, 2005) which are synthesized on different FPGA devices. It shows that,

Table 1: MAC performances comparison with different pipeline stages on stratix 3

	Frequency (MHZ)		Area (LUTs)	
Pipeline stages	Paidimarri *et al.* (2009)	Proposed MAC	Paidimarri *et al.* (2009)	Proposed MAC
7	-	193.69	-	1479
9	221	262.95	1890	1559
11	234	265.67	3141	1590

--: Not listed, MAC: Multipler accumulator, LUT: Look-up-table

Table 2: MAC performances comparison on different devices

Design	Devices	Pipeline stages	Frequency (MHZ)	Area (LUTs)
Proposed MAC	Stratix III	9	262.95	1559
Jin *et al.* (2006)	Virtex IV	13	123.60	1774
Dou *et al.* (2005)	Virtex II	13	200.00	2184

MAC: Multipler accumulator, LUT: Look-up-table

Table 3: Matrix multiplication performance comparison

Design	No. of PEs	Frequency (MHZ)	Area (LUTs)	DSPs	Block memory (9kb)
Proposed matrix multiplier	10	179.2	15600	40	4927
	20	178.3	30846	80	5120
Zhuo and Prasanna (2007)	10	105.0	41157	-	160
Fovanovic and Milutinovic (2012)	252	161.0	290556	2016	4032

--: Not listed, PE: Processing element, LUT: Look-up-table and DSP: Digital signal processing

with lower pipeline stages, the proposed MAC still can achieve higher frequency and less resources consumption.

The above experimental results show that the MAC proposed in this study has a trade-off between frequency and resources consumption and its performance is superior to others.

Performance of matrix multiplication: From Table 1, it is noticed that the frequency and area consumption is better than others when the pipeline stage is 9. So, the matrix multiplication is designed using this pipeline stage directly. With different number of PE, the requirement of hardware resources increases linearly. The dimension of matrices A and B also can be configured in the matrix multiplication. Table 3 shows the detailed computing performance of the matrix multiplication when the dimension of matrices A and B are 30×40 and 40×30, respectively. In the table, Block Memory is M9k and the unit memory size of it is 9 kb. In the design of Zhuo and Prasanna (2007) and (Fovanovic and Milutinovic, 2012), their Block Memory consumption includes only the computing part of the matrix multiplication which doesn't involve the storage of the matrices data and the memory consumption of the storage increases linearly with the matrix dimension. In this study, the memory consumption includes both storage and computation memory. For the computation part, the consumption can be obtained by subtracting Block Memory size of 20 PEs' and 10 PEs'. So the memory consumption for 10 PEs is 5120-4927 = 193. Then the remaining size is the storage consumption which is 4927-193 = 4734 for 10 PEs. Since, number of LUTs and DSPs is proportional to that of PEs', the consumption of LUT and DSP of each PE can be calculated by dividing the total consumption by the number of corresponding PE.

Table 4 represents the resource consumption of each PE. As shown in the table, the proposed matrix multiplier consumes 18.75% more Block Memory than Zhuo and Prasanna (2007) and Fovanovic and Milutinovic (2012)

Table 4: Resources consumption of each PE in matrix multiplication

Design	Area (LUTs)	DSPs	Block memory (9kb)
Proposed matrix multiplier	1560	4	19
Zhuo and Prasanna (2007)	4116	-	16
Fovanovic and Milutinovic (2012)	1153	8	16

--: Not listed, LUT: Look-up-table and DSP: Digital signal processing

and it consumes 35.3% more LUTs than (Fovanovic and Milutinovic, 2012). Despite of that fact, the proposed matrix multiplier consumes only 37.9% LUTs of Zhuo and Prasanna (2007), half of DSP resources of Fovanovic and Milutinovic, 2012 and achieve higher frequency than both of them.

Peak performance is another popular metric for floating-point performance. As the matrix multiplication designed in this study can complete one floating-point multiplication and one floating-point addition in one clock cycle, so the peak performance can be written as:

$$\text{PERF} = 2\times P\times F_{req}$$

where, P is the number of PE in the matrix multiplication, F_{req} is the working frequency. When the number of PE is 10, the peak performance of the matrix multiplication designed in this study is 3600 MFLOPS. When the number of PE is 20, it can achieve 7200 MFLOPS.

In the proposed MAC architecture, there is an addition operation of the last N-stage pipeline data. As the last N-stage data from pipeline adders have to be stored to registers, for small matrices, it needs to wait each PE completing calculation to avoid conflicts in the registers' data and the waiting time is non-ignorable compared to the overall time. But when comes to large matrices, the time of multiply accumulation is long enough for the last addition to fetch data from registers, so data from memory can be fed into the MAC consecutively. In this case, it only needs to wait for the last N-stage after all PEs completing calculation. Compared to the overall computing time, such time is so short that it can be ignored. Assuming the dimension of the matrices A and

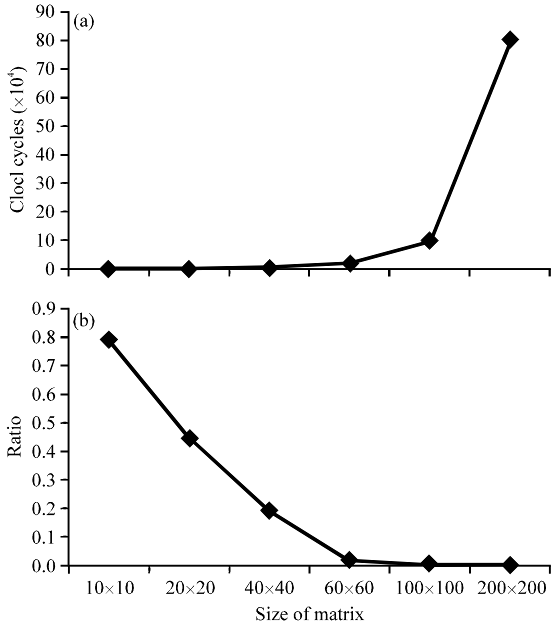

Fig. 6(a-b): (a) Consumption of clock cycles with different matrix size in MAC and (b) Ratio of waiting time to the whole clock-cycle consumption with different matrix size in MAC, MAC: Multiplier accumulator

B are m×n and n×*l*, respectively, clock cycles for computing are proportional to m×n×*l*/10. If the dimension of matrices A and B is equal, Fig. 6a and b show the clock cycles the computing process of the matrix multiplication consumes and the ratio of the waiting time to the whole consumed clock cycles, respectively with 10 PEs. In the Fig. 6a, the clock cycles are 46, 1440, 6438, 10038 and 80038, when the sizes of matrices are 10×10, 20×20, 40×40, 100×100 and 200×200, respectively. So with the size of matrices increases, clock cycles of the computing require increase following a cubic curve. However, as the computing time of the last N-stage is fixed, the ratio of the waiting time duration to the overall clock cycles is getting smaller and smaller. It finally goes to 0.00037 and 0.000048 when the size of matrix goes to 100×100 and 200×200. Thus, the matrix multiplier designed in this paper is more suitable for large scale matrices.

CONCLUSION

This study proposes a novel architecture for pipelined MAC and implements matrix multiplier based on the proposed MAC. The experiment results demonstrate that the proposed architecture can effectively improve the performance of the MAC and the matrix multiplication. The number of PE and the dimension of matrices are configurable in matrix multiplier, so the matrices can be easily extended to any size. And the efficiency of the matrix multiplier is even higher when the matrices are larger. In the future work, designing a more efficient IP core of MAC is necessary for improving the computing performance of matrix multiplier.

ACKNOWLEDGMENTS

This study is supported by Scientific Innovation Research Foundation of Hunan Province of China (Grant No. 2010TT1002), Fundamental Research Funds for the Central Universities (Grant No. 531107040421) and Natural Science Foundation of China (Grant No. 60973030).

REFERENCES

Amira, A., A. Bouridane, P. Milligan and P. Sage, 2000. A high throughput FPGA implementation of a bit-level matrix product. Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems, August 8-11, 2000, Lansing, Michigan, pp: 396-399.

Beauchamp, M.J., S. Hauck, K.D. Underwood and K.S. Hemmert, 2008. Architectural modifications to enhance the floating-point performance of FPGAs. IEEE Trans. Large Scale Integr. Syst., 16: 177-187.

Campbell, S.J. and S.P. Khatri, 2006. Resource and delay efficient matrix multiplication using newer FPGA devices. Proceedings of 16th ACM Great Lakes Symposium on VLSI, April 30-May 02, 2006, New York, USA, pp: 308-311.

Dou, Y., S. Vassiliadis, G.K. Kuzmanov and G.N. Gaydadjiev, 2005. 64-bit floating-point FPGA matrix multiplication. Proceedings of the 13th ACM International Symposium on Field Programmable Gate Arrays, February 20-22, 2005, Monterey, California, USA., pp: 86-95.

Fovanovic, Z. and V. Milutinovic, 2012. FPGA accelerator for floating-point matrix multiplication. IET Comput. Digital Tech., 6: 249-256.

Jang, J.W., S.B. Choi, V. K. Prasanna, 2005. Energy and time efficient matrix multiplication on FPGAs. IEEE Trans. Very Large Scale Integr. Syst., 13: 1305-1319.

Jin, X., X.P. Gao and X. Long, 2006. Floating-point multiply-accumulative processing element on FPGAs. Comput. Digital Eng., 34: 165-168.

Liu, P.H., H.X. Lu, G.L. Gong and W.P. Liu, 2012. Design of an FPGA based double precision floating point matrix multiplier with pipeline architecture. CAAI Trans. Intell. Syst., 7: 302-306.

Luo, Z. and M. Martonosi, 2000. Accelerating pipelined integer and floating-point accumulations in configurable hardware with delayed addition techniques. IEEE Trans. Comput., 49: 208-218.

Paidimarri, A., A. Cevrero, P. Brisk and P. Ienne, 2009. FPGA implementation of a single-precision floating-point multiply-accumulator with single-cycle accumulation. Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines, April 5-7, 2009, Karachi, Pakistan, pp: 267-270.

Qasim, S.M., S.A. Abbasi and B. Almashary, 2008. A proposed FPGA-based parallel architecture for matrix multiplication. Proceeding of the 2008 IEEE Asia Pacific Conference on Circuit and Systems, November 30-December 3, 2008, Macao, China, pp: 1763-1766.

Riaz, K., M.S.H. Khiyal and M. Arshad, 2005. Matrix equality: An application of graph isomorphism. Inform. Technol. J., 4: 6-10.

Tian, X., F. Zhou, Y. W. Chen, L. Liu and Y. Chen, 2008. Design of field programmable gate array based real-time double-precision floating-point matrix multiplier. J. Zhejiang Univ. Eng. Sci., 42: 1611-1615.

Xu, F., Y. Xi, C. Hong and J. Weiwei, 2011. Design and implementation of matrix hardware acceleration based on FPGA/Nios II. J. Electron. Meas. Instrum., 25: 377-383.

Yang, C.N., L.H. Ma, Y.Z. Yue and H. Li, 2012. A real-time implementation scheme of attitude and velocity algorithms in Sins. Inform. Technol. J., 11: 1650-1654.

Zhu, S.H., 2013. Hardware Implementation Based on FPGA of interrupt management in a real-time operating system. Inform. Technol. J., 12: 943-950.

Zhuo, L. and V. K. Prasanna, 2007. Scalable and modular algorithms for floating-point matrix multiplication on reconfigurable computing systems. IEEE Trans. Parallel Distrib. Syst., 18: 433-448.