International Journal of Engineering Research and General Science Volume 5, Issue 3, May-June, 2017
ISSN 2091-2730

Efficient Architecture for Matrix Multiplication using Floating Point Multiplier

Y.R.Annie Bessant[1], S.Monisha [2]
[1, 2] Department of Electronics and Communication Engineering,
[1, 2] St.Xavier's Catholic College of Engineering, Nagercoil, India
[1]annieben08@gmail.com [2]monisha.sathish1993@gmail.com

Abstract- In this paper, a new area and delay minimization architecture is proposed for rank-1 update matrix-matrix multiplication whose inputs are double precision floating point number. The design is based on pipelined multiplication which handles matrix of arbitrary sizes; the processing datas are stored in dedicated on-chip BRAM. This minimization is introduced to designers as a trade-off between bandwidth and local memory. Analysis is presented for the design parameters optimal choice. The hardware architecture is described in Verilog HDL synthesized for a family virtex6 and device SX240T FPGA, which scale more than 40 processing elements. Various parameters like LUTs, Slices, bonded IOBs, frequency, DSP48E1S, delay, Power, CPU Completion time and Memory usage are analysed. Rank1 update consumes a power of 0.326watt and has a delay of 7.314 ns respectively. Comparing other rank-1 multiplication methods our proposed algorithms uses 15% less area resources and improves the delay in 12%.
Keywords— Computation efficiency, Field Programmable Gate Array, Floating point arithmetic, Matrix, System performance, Pipeline Architecture,Processing Element.

INTRODUCTION

Many systematic computing applications use floating point multiplication as a basic maneuver and it require more hardware resources for its implementation on Field Programmable Gate Array (FPGA). A set of rule in basic linear algebra subprograms is known as BLAS. It provides essential structure for many linear algebra applications such as vector, Eigen value, linear equations, and matrix operations. The complete study of Level-3 BLAS provides matrix-matrix multiplication. The linear array architecture [8, 9,10] obtains peak performance by memory switching and a modular processing element [10] improves the performance to 8.3GFLOPs .When the number of PE increased the clock degradation was 15%.Algorithm implemented on Cray XDI with XC2VP50 FPGA [9] reduces the routing complexity by utilizing the available resources. The performance estimated as 2.1 GFLOPS of 130 MHZ and speed degradation was 35%.Pipelined floating point MAC [1], which works on square matrix and does not support zero and other denormal numbers. The performance obtained was 15.6GFLOPS on Virtex-II proxc2v125 with frequency of 200MHZ and memory utilization of 125 Mb. Broadcast matrix element to all PEs [5, 6] reuse of that data and overlapping IO obtain sustained performance of 29.8 GFLOPs. This paper, we design architecture for double precision floating point matrix multiplication based on rank-1 update scheme target at family virtex5 and device SX240T FPGA. The main aim is to reduce the area and delay .The processing element for the first algorithm is based on pipeline processing, where the input data are fetched and executed. By reusing the input data and full utilization of PE the peak performance is obtained. This paper is well ordered as follows. In section II the proposed algorithm is explained. The result is discussed in section III and the conclusion in section IV.

MATRIX MULTIPLICATION ARCHITECTURE

Matrix multiplication algorithm for a rank-1 update is of the form $C=\alpha A^T B +\beta C$ in this case $\alpha, \beta =1$.Consider three matrixes A, B, C of dimension P×Q, Q×R and R×S, respectively. When two matrices of sizes $S_a$andS_b are multiplied, the result we obtained is $S_a \times S_b$.The goal of this algorithm is complete utilization of resources and concatenation of I/O and computation and hence improves performance. This goal is attained by high bandwidth cost and use of complete available resources.Algorithm1 implicit $S_a \geq S_b = P$ and $S_b = P$ elements of B were being re-used S_a periods .This design confirm more reprocess by permitting S_a and S_b to be larger than P and S_b to be multiple of P, Where P be the number of processing element.

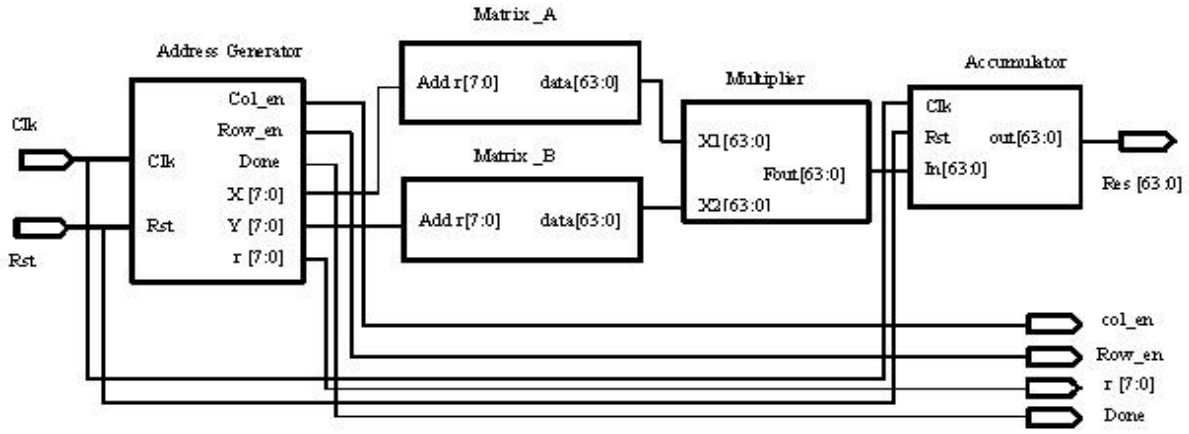


Figure 1. Register Transfer Level Schematic for Rank-1 update scheme

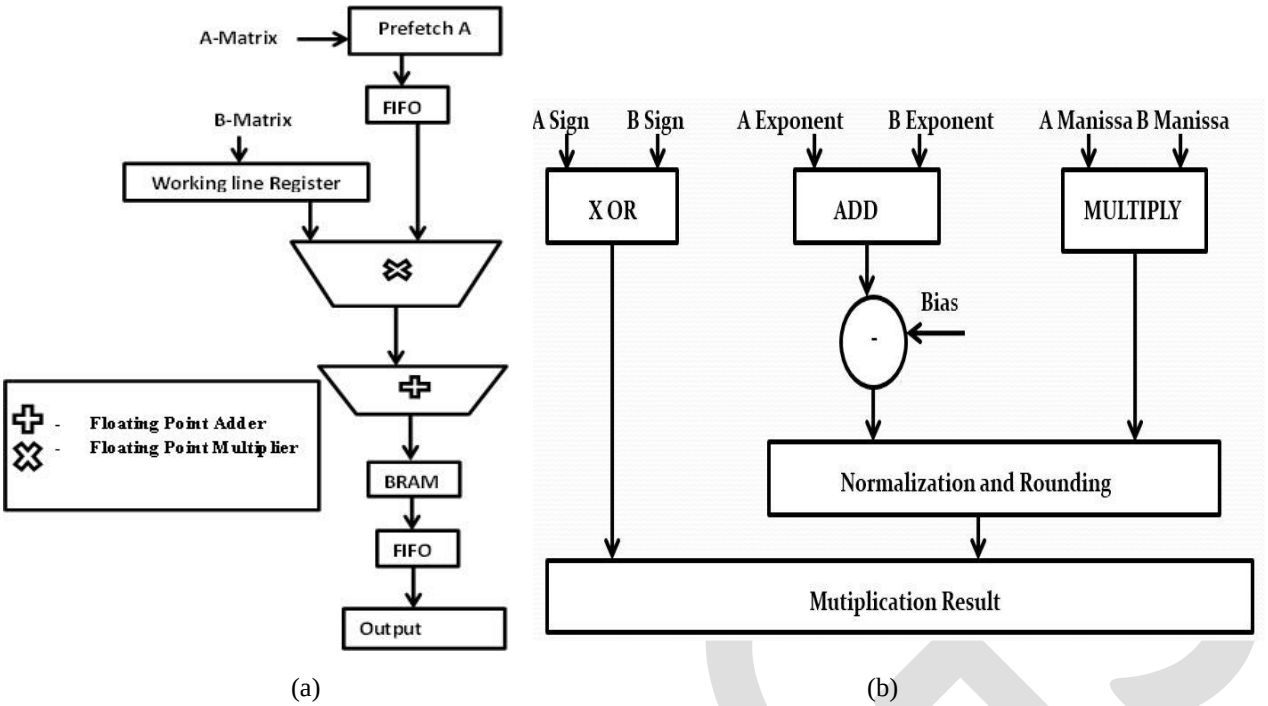
Component description

Figure 1shows the different components involve in the proposed architecture.

1) A-Matrix: This register is used to store the elements of A- matrix. The input to this register is from the address generator and the floating point multiplier reads the elements from matrix A to perform multiplication.
2) B-Matrix: This register is used to store the elements of B-matrix. The input for the register is from address generator and multiplier collects the element of matrix B to multiply with matrix A.
3) Multiplier: The data (either A-Matrix or B-Matrix) received by the floating point multiplier is 64 bits. In this block a standard floating point double precision multiplier is used. The output of floating point multiplier is given to the accumulator block.
4) Accumulator: It is used to store the product. It consists of three input one from the floating point multiplier other from the address generator and a clock signal. The accumulator consists of fdr (fast dump restore) and addf, the addf is used to add each product with the past product.
5) Address generator: Processing element consists of an execution unit known as address generator. It is used to calculate addresses in order to improve the memory access. The number of CPU cycles required for the execution of floating point matrix multiplication is reduced by this address generator as it works in parallel to rest of the PE. It consists of clock and reset. Clock is used to perform multiplication per cycle. And reset is used to eliminate the present element in order to process the next multiplication.

Data flow

Figure 2a explains the overview of the processing element operation. The element from the matrix A and matrix B is fed in the multiplier in terms of column major and row major respectively. The first row of matrix A is fed in to the fetch register. After the complete loading of the element the row is fed into the working line of register. During this time the next row of the matrix A is fetched by the fetch register. Meantime element from matrix B is broadcast into the multiplier and the output is processed and stored in the accumulator. As the data does not depend on the current multiplication, stalling can be avoided in this multiplier. Due to this, there is no interruption in data processing. Thus this process continues until all the elements are processed and the final output is stored .The location of element is calculated by the address generator and stored in the particular place. The merit of this design is complete overlap of I/O and computation. Therefore all PEs are in operation only exception is during latency period. No interruption is noticed in this design. High performance and less execution time which indicate the speed.



(a) (b)

Figure 2(a).overview of proposed architecture. (b) Structure of Double Precision Floating point Multiplication

A Double Precision Floating Point Multiplication

Multiplication of double precision floating point numbers is carried out field wise. The given values are first converted into 64 bit binary number(1bit sign,11 bit manitissa,52 bit exponent),after conversion multiplication is carried out for each fields separately as shown in Figure 2b.

Sign calculation*:* The Sign of the number to be multiplied is obtained .If the sign is negative it indicates one and if the number is positive it indicates zero. It is added and complement is taken. For this purpose XOR gate is used. The product is positive when the two operands have the same sign, otherwise it is negative.

Sign,S= Sa XOR Sb

***Exponent calculation*:** It is carried out by adding the exponent of number to be multiplied followed by subtracting the added value with bias. The bias value is 1023 for double precision floating point numbers. The eleven bit in the exponent represents a biased exponent, which is obtained by adding 1023to the actual exponent. The principle of the bias is to lethuge or very small numbers without requiring a separate sign bit for the exponents. The biased exponents allow a range of actual exponent values from -1022 to +1023.

***Mantissa calculation*:**. The 52 bit mantissas of the two numbers are multiplied and subjected to normalization and rounding. The mantissa bit of multiplication after, the normalization is applied to final resultant for bring back the 64-bit format.(i.e.) sign-exponent-mantissa.

DESIGN EVALUATION

In this section, the FPGA implementation of our algorithms is discussed. Our target device is Xilinx virtex-5 SX240T FPGA and we use Xilinx ISE13.5 as simulation tool. We examine the performance of our algorithms using different parameters like frequency, delay, power, memory, number of slices used and bonded IOB. Figure3 and figure 4 shows the output wave form and technology schematic for floating point multiplication

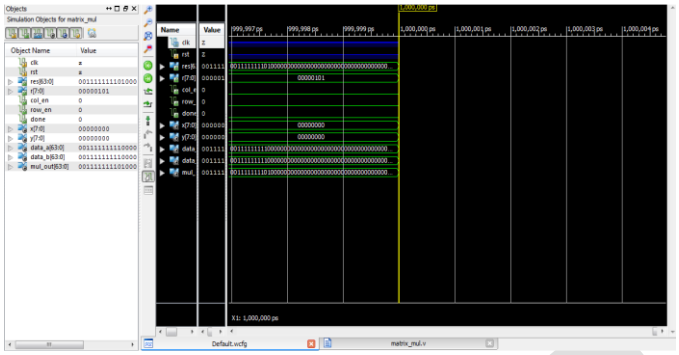


Figure 3 Output Waveform for Double Precision Floating point Matrix multiplication

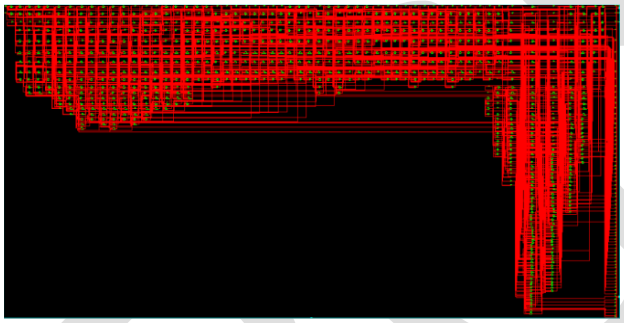


Figure 4. Technology Schematic for floating point matrix multiplication

. Table 1 gives the performance analysis of our algorithm . Table 2 explains the performances analysis of Architectures on different FPGAs.Pipelined architecture consumes less power and delay than other architecture .The memory usage is 50% less for some devices , the delay is 0.75% less and the power consumed is also 0.75% less when compared with other architecture. Due to abundance of available resources and proper pipelining, our architecture accommodate more number of PEs (more than 40).On comparing our algorithms with rank 1 update scheme,as reported [10,11] by scaling 20 PEs the degradation in frequency by 35% and 15%,in our design we scale more than 40 PEs with negligible degradation in frequency. It consumes 8%less delay than rank 1 update scheme. Design requires square matrices [1, 10], our algorithm support matrix of arbitrary sizes. Using intermediate term partaking the area reduced to 10% when compared with [1, 4, and 9]. Our floating point MAC unit support zeros and abnormal numbers [1] doesn't hold the above. For Computation and overlapping I/O our algorithm used pipeline processing in resist to memory switching [1,10]. Our design achieves the peak performances by using 90% of DSP blocks, 202.416Mbyte of memory and frequency of 189.517MHZ.

TABLE 1. PERFORMANCE ANALYSIS OF ARCHITECTURES ON DIFFERENT DEVICES.

Pipelining Architecture				
DEVICES	Power (W)	Frequency (MHZ)	Delay (ns)	Memory (M Byte)
VIRTEX 4	0.176	118.23	8.456	220.68
VIRTEX 5	0.382	131.89	7.582	224.648
VIRTEX 6	2.520	189.517	5.277	202.416
VIRTEX 7	-	231.722	4.3162	231.856

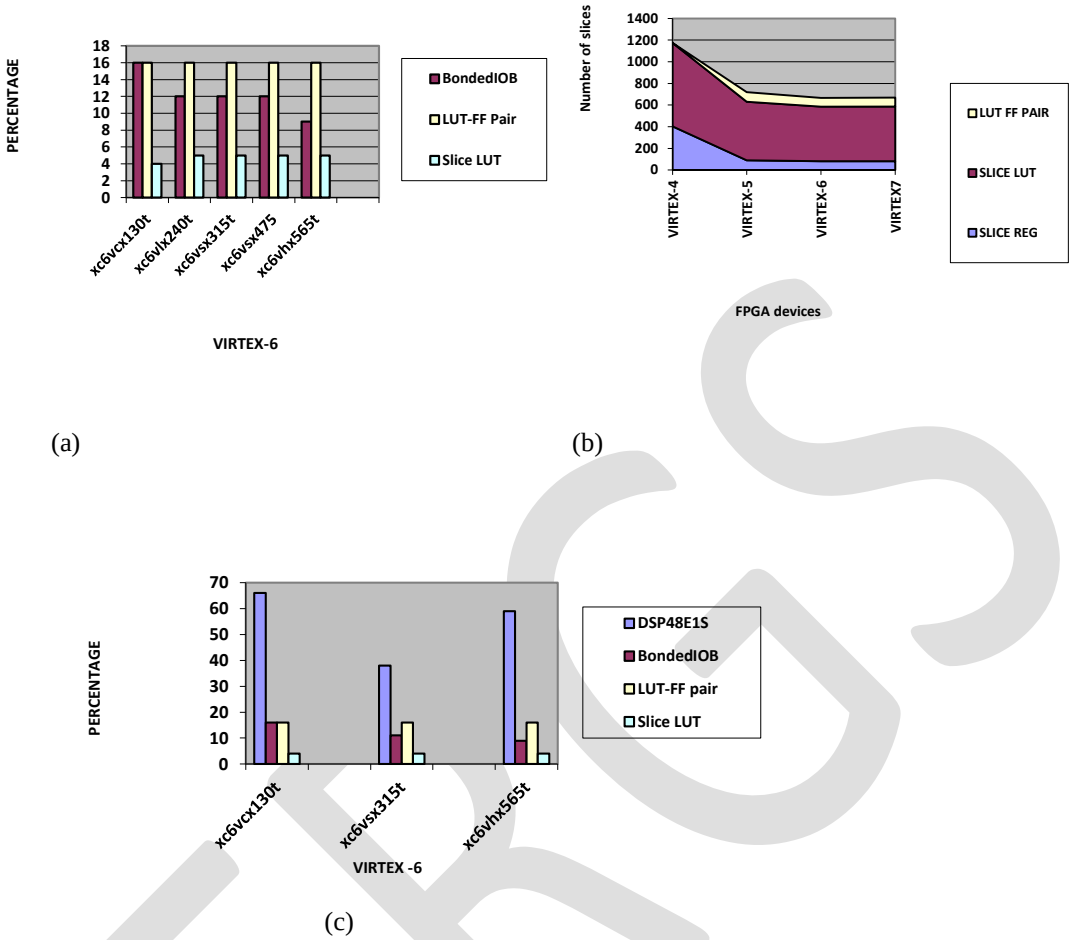


Figure 3. Comparison of various parameters on (a) Different devices for pipelined multiplication. (b) Different virtex family for floating point multiplication.

CONCLUSION

In this paper architecture for floating point matrix multiplication was presented with trade-off between memory and frequency. Multipliers are the foremost segment in area and delay consumption for matrix multiplication. In the proposed architecture floating point multiplier was used. The performance analysis shows that the performance of our algorithm improved in term of delay and area. The algorithms are able to sustain peak performance with a design frequency of 189.517MHZ on virtex-5 SX240T FPGA.

REFERENCES:

[1] Y.Dou, S. Vassiliadis ,G. K Kuzmanov, and G.N Gaydadjiev"64-bit Floating-Point FPGA Matrix Multiplication"Proc.13thACM/SIGDA International Symposium Field-Programmable Gate Arrays (FPGA), ISBN-number-1-59593-029-9, pp. 86-95,February 20-22,2005.

[2] Jang J.W., Choi S. and Prasanna V.K. "Area and Time Efficient Implementation of Matrix Multiplication on FPGAs" Proc.first IEEE international conference Field Programmable Technology, ISBN-number-0-7803-7574-2,December 16-18,2002

[3] Ju-Wook Jang, Seonil B. Choi,, and Viktor K. Prasanna "Energy- and Time-Efficient Matrix Multiplication on FPGAs" IEEE Transaction on a Very Large Scale (VLSI) System, vol. 13, no. 1,ISSN-number-1063-8210,pp.1305-1319,November 2005.

[4] Kumar, V. B. Y., Joshi, S., Patkar, S. B., and Narayanan, H "FPGA Based High Performance Double Precision Matrix Multiplication ", International Journal Parallel Programming, Vol.13,Issue 3,pp. 322-338,June 2010.

[5] Ling Zhou and Prassana V.K "High- Performance Designs for Linear Algebra Operations on Reconfigurable Hardware" , IEEE transaction on Computers ,Vol. 57, Issue.8,ISSN-number-0018-9340,pp.1054-1071,August 2008.

[6] K.D. Underwood and K.S. Hemmert, "Closing the Gap: CPU and FPGA Trends in Sustainable Floating-Point BLAS Performance,"Proc.12th Ann.EEE Symp.on Field-Programmable Custom Computing Machine,ISBN number-0-7695-2230-0,April 20-23,2004.

[7] Sneha Khobragade, Mayur Dhati "Review on Floating Point Multiplier Using Vedic Mathematics". International Journal of Science and Research (IJSR), Vol.4, Issue 2, ISSN-number-2319-7064,pp.1498-1502,February 2015.

[8] Ting Zhang, Cheng XuYunchuan Qin and Min Nie "An Optimized Floating Point Matrix Multiplication on FPGA", Informational Technology Journal,Vol.12, No.9, pp.1832-1838,July 3,2013.

[9] Zhuo, L., and Prasanna, V. K. "Scalable and Modular Algorithms for Floating Point Matrix Multiplication on Reconfigurable Computing Systems", IEEE Transaction on Parallel Distribution System, Vol.18, No4, pp. 433-448,April 2007.

[10] ZhuoL.,Morris G.R and Prasanna "High-Performance Reduction Circuits Using Deeply Pipelined Operators on FPGAs" IEEE Transaction on Parallel and Distributed System ,Volume:18 , Issue:10,ISSN-1045-9215,September 17,2007.

[11] Zhuo L. and Prasanna V. K. "Scalable and modular algorithms for floating-point matrix multiplication FPGAs", IEEE Transaction On Proceedings of 18th IPDPS, page 92.

[12] Sonawane D.N, Sutaone M.S, Inayak Male "Resource Efficient 64-bit Floating Point Matrix Multiplication Algorithm using FPGA", Proc.TENCON-IEEE Region 10 Conference, Sanfrancisco, ISMN-978-1-4244-4546-2,January 23-26, 2009,