Simulation & Synthesis of FPGA Based & Resource Efficient Matrix Coprocessor Architecture

Jai Prakash Mishra[1], Mukesh Maheshwari[2]

[1]*M.Tech Scholar, Electronics & Communication Engineering, JNU Jaipur, Rajasthan, India*

[2]*Assistant Professor, Electronics & Communication Engineering, JNU Jaipur, Rajasthan, India*

Abstract

Due to fast routing and logic resources, embedded multipliers, very high gate densities, block RAM memory modules, and embedded soft processors, modern FPGAs offer an excellent platform for the implementation of efficient Field Programmable Systems on Chip. The latter are largely required in a wide range of applications, like telecommunication, wireless, networking, video, signal processing, robotics and digital control. All the above applications are characterized by an intensive computation of matrix operations like the matrix multiplication. Therefore, the introduction of a coprocessor to support the computation of matrix multiplications alongside a general purpose processor system can be a good solution to reduce the computational time & Resources. However, in order to get the most of their potential performance, FPGAs need to be programmed at the hardware low level and not the application level. This process needs a considerable hardware knowledge, which means that FPGA programming is still reserved to the specialist. The recent explosion in FPGAs resource densities and performance, together with their inherent reprogram ability feature, makes them very attractive as high performance, flexible, implementation platforms for these operations. In this present work a novel architecture has been developed for large matrix multiplication. For high performance applications, this operation will minimize the hardware resources. For this, we use a parallel architecture for the multiplication of two matrices using Field programmable Gate Array (FPGA).

Index Terms: *FPGA, Xlinx ISE, Modelsim, Matrix Multiplication, Parallel Block-Scheduling.*

1. INTRODUCTION

Matrix Multiplication Architecture is a processor intended to perform matrix multiplication in an efficient manner, with reduced consumption of time and resources. It finds enormous use in real time image processing and computer vision applications, where the images are considered as matrices, and involve various matrix operations of which multiply is highly important and relatively difficult to implement. Matrix multiplication is a core operation in digital signal processing operations with a variety of applications such as image processing, computer graphics, sonar processing and robotics [2][3][4]. For high performance applications, this operation must be realizes in hardware. For this, we use a parallel architecture for the multiplication of two matrices using Field programmable Gate Array (FPGA) [1].

Traditionally, matrix multiplication operation is either realized as software running on fast processors or on dedicated hardware such as Application Specific Integrated Circuits (ASICs). Software based matrix multiplication is slow and can often become a bottle-neck in the overall system operation. However, hardware (Field Programmable Gate Array (FPGA)) based design of matrix multiplier provides a significant speed-up in computation time and flexibility as compared to software and ASIC based approaches respectively [6].

Modern FPGAs can accommodate multimillion gates on a single chip. During the last decade, the logic density, functionality and speed of FPGA have

improved considerably. Modern FPGAs are now capable of running at speed beyond 500 MHz [7]. Another important Feature of FPGAs is their potential for dynamic reconfiguration [8]; that is, reprogramming part of the device at run time so that resources can be reused through time multiplexing.

The matrix multiplication operation is often performed by parallel processing systems which distribute computations over several processors to achieve significant speedup gains. There exists many realization of matrix multiplication. These realizations mainly differ in terms of algorithms or the hardware platforms.

In this work, we consider the problem of implementing large matrix multiplication algorithm [10] and then mapping the algorithm to parallel architecture on FPGA. The FPGA-based systolic array parallel architecture for the tri-matrix multiplication was evaluated for different matrix size [9], but if the size of tri-matrix was increased then it required more hardware resources which were the computational complexity of multiplier. Since this design could not be fit into Spartan-3 and large size matrices requires more hardware resources, more memory space, more power, RAM and more time requires for complete the computation. For matrix multiplication of large matrices, the memory based architecture is quite efficient whereas, for small and medium sized matrix multiplication.

2. The PARALLEL BLOCK (PB) Architecture

2.1 Algorithm

We separate the sequential block algorithm into two parallel algorithms, called Master and Slave. The Master algorithm is executed on a single processor, and the Slave on multiple processors. In the discussion to follow and Shown in fig-1, we refer to the Slave processors as to Processing Elements (PE). The Master sends the data from matrices A and B within messages and loads the results into matrix C ordered as *Si* by *Sj* blocks, according to the Parallel Block scheduling algorithm [1]. The data in the messages are correctly ordered by the Master and are delivered in a preserved order to the PE chain. In each PE, the scheduling algorithm is performed in the following steps:

Step 1: The Master processor sends *Si* elements of one column of array A so that each PE receives *Si/P* elements.

Step 2: The Master processor sends *Sj* elements of

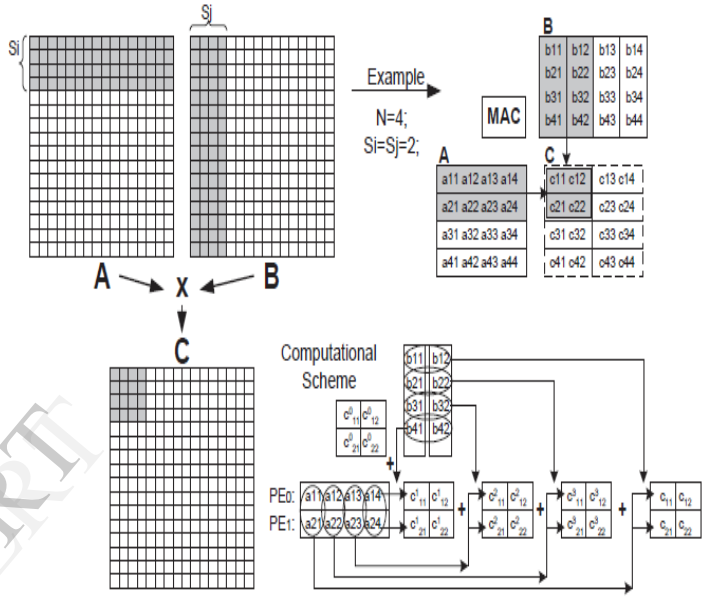

Fig-1 The PB computational scheme – an example for N = 4 and $S_i = S_j = 2$

One row of array B to all PEs. The elements of array A and B are multiplied in each PE and added to the corresponding temporary elements of array C. Results are accumulated into the local PE memory.

Step 3: Repeat N times steps 1 and 2. Finally, the PE local memories will contain *Si* × *Sj* elements of C.

Step 4: The Master processor transfers the *Si* × *Sj* block of C from the PE local memories to the main memory. If there are unprocessed blocks, go to step 1.

2.2 Explanation

The Main Processor (comprises of Matrix A, Matrix B
and Matrix C) sends a signal for data transfer from Matrix A and Matrix B which is received by Slave processor (comprises of MAC unit) in the form

of data acknowledgement signal.

As the Slave receives the data acknowledgement signal it fetch the data from Matrix A and Matrix B and store it in respective FIFO registers.

As per the functionality described above regarding the parallel block architecture, the data elements are fed to register 1 to register N, depending upon the size of matrix. If the size of matrix is 8x8 then the PEs required is 4. Then the data are multiplied and there after addition is performed, the resulted output is stored in output register from C_{11}_reg to C_{41}_reg. The computed result is fed to Matrix C.

The basic building block of proposed matrix multiplication is shown in figure-2. This building block comprises of Matrix A, Matrix B, MAC Unit and Matrix C.

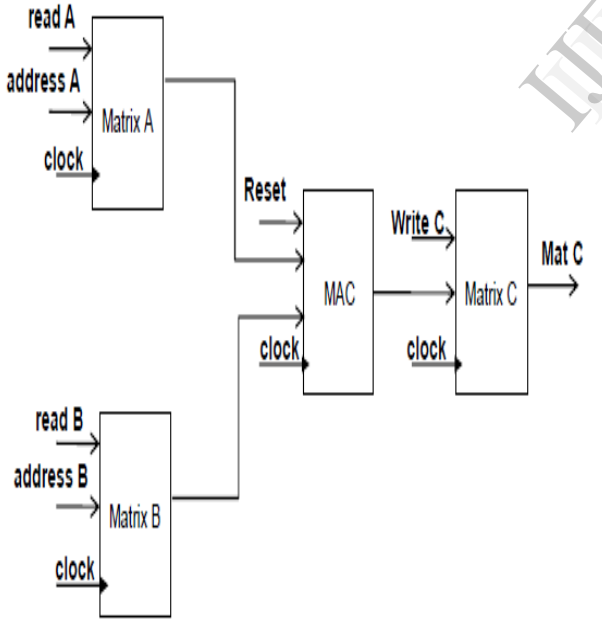

Fig-2 Basic block building architecture

Matrix-A

Matrix A is having elements of *I* x *K* matrix structure. The length of elements is $2^n$where n is in the multiples of 2. A clock signal is provided to it. When the read A signal is high, Slave processor will fetch data from Matrix A. The address A signal will increment in the multiples of 2.

Matrix-B

Matrix B is having elements of *K* x *J* matrix structure. The length of elements is 2^n where n is in the multiples of 2. A clock signal is provided to it. When the read B signal is high, Slave processor will fetch data from Matrix B. The address B signal will increment in the multiples of 2.

MAC Unit

Basically MAC Unit comprised of Slave Processor which performs all the arithmetic operations. In Slave Processor individual processing elements are there which depends upon the length of matrix. When the reset signal is active high, the Slave Processor comes to reset hence before starting any new task/operation the reset signal should be kept high for one clock period.

Matrix C

Matrix C is of I x J matrix structure. After the results being computed from MAC Unit, Shown in fig-3. The data is stored in Matrix C according to the address location.

The input signal to both the PE and Slave Processor is same whereas the output of the PE is C1, C2, C3 and C4 whereas the output of the Slave Processor is controlled by 2:1 multiplexer. Data Acknowledgement and Data from Main are the two important signals of Slave Processor Shown in fig-3. Until the Data acknowledgement signal is low; FIFO A and FIFO B will not fetch the data from respective matrices. Similarly when Done from Main signal becomes active high it means that all the required data has been fetched.

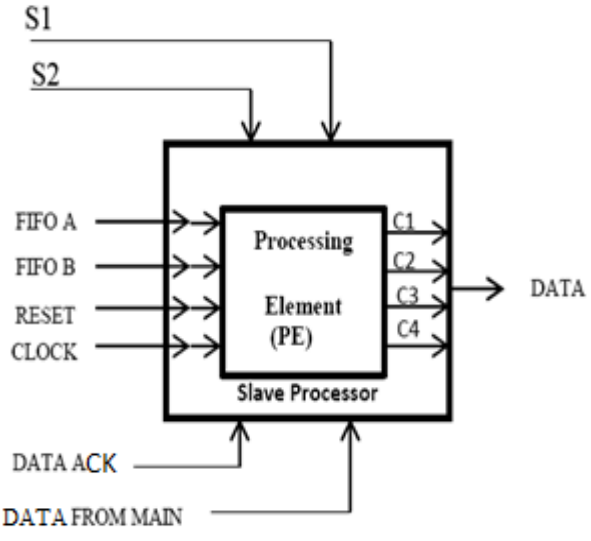

Fig-3 Basic MAC Unit

2.3 Algorithm for Main Processor

1) Reset Main
Reset all counter and reset slave processor.

2) Main 1

Read 32 bit from Matrix A and Matrix B; increment address A count and address B count by 2.

3) Main 2
Load data from Matrix A Out to FIFO A and Matrix B Out to FIFO B; send data acknowledgement to Slave Processor; increment N count.

4) Main 3
Check N count; if N count=7 then go to main 4 else go to check for data request.

5) Check for data request
If data request =1 then go to Main1 else remain in check for data request.

6) Main 4
Send done from main signal to Slave Processor; reset N count and increase final count by 1.

7) Main 5
Load Ist set of data from Slave to Matrix C of Main; go to Main 6.

8) Main 6
Load IInd set of data from Slave to Matrix C of Main; go to Main 7.

9) Main 7
Load IIIrd set of data from Slave to Matrix C of Main; go to Main 8.

10) Main 8
Load IVth set of data from Slave to Matrix C of Main; go to Main 9.

11) Main 9
If final count=1 then go to Main 10.

If final count=2 then go to Main 11.

If final count=3 then go to Main 12.

If final count=4 then go to Reset Main.

12) Main 10
Set reset address A count and set preset address B count; go to Main 1.

13) Main 11
Set preset address A count and set preset address B count; go to Main 1.

14) Main 12
Set preset address A count and set preset address B count; go to Main 1.

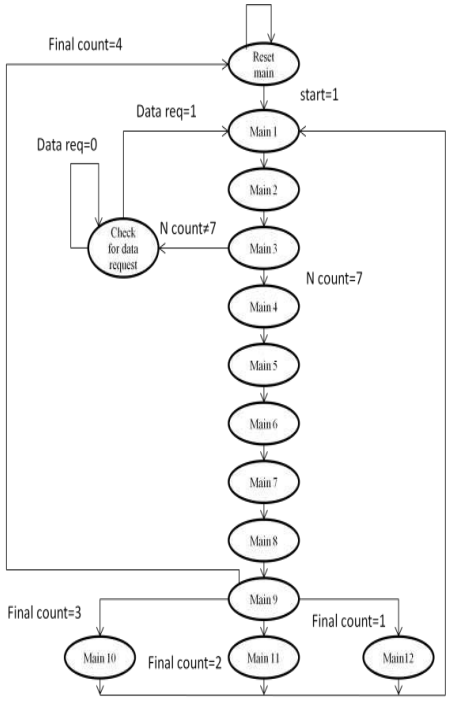

Fig-4 FSM of Main Controller

2.4 Algorithm for Slave Processor

1) Check for data ack

Wait for data ack from the main processor.

2) Slave 1

1 cycle is required to store data in registers from FIFO A and FIFO B.

3) Slave 2

1 cycle is required to compute results; wait for done from main; from the main processor if done from main = 0 then go to check for data ack else go to slave 3.

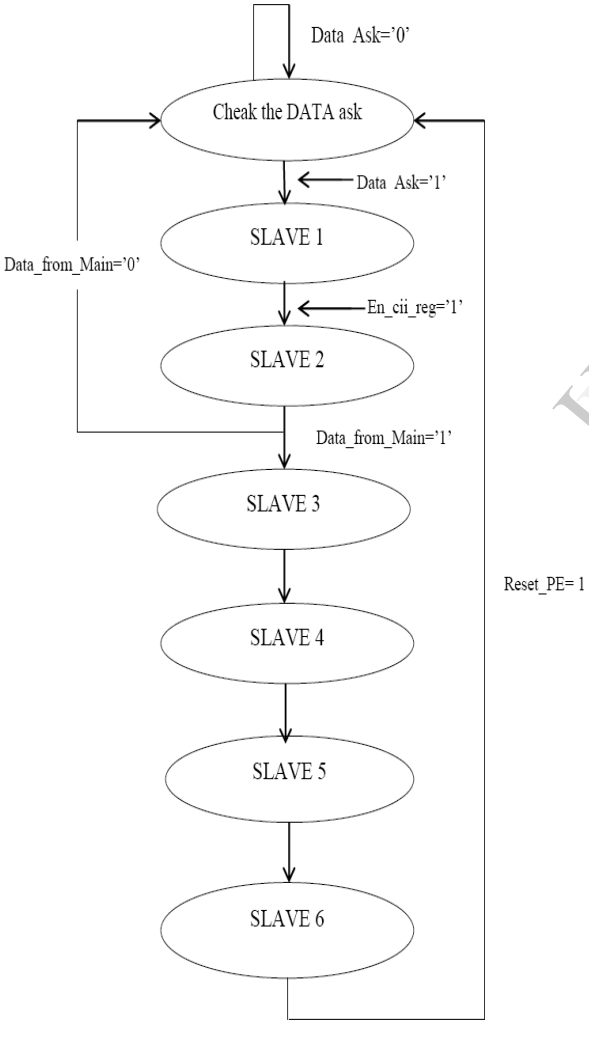

Fig-5 FSM for Slave Controller

4) Slave 3

1 cycle is required to send I set of data to main processor.

5) Slave 4

1 cycle is required to send II set of data to main processor.

6) Slave 5

1 cycle is required to send III set of data to main processor.

7) Slave 6

1 cycle is required to send IV set of data to main processor; also data req = 0 is sent to main processor.

3. SIMULATION RESULTS

The matrix multiplication Architecture is simulated by using Xilinx 10.1i ISE Simulator by writing VHDL hardware description language.

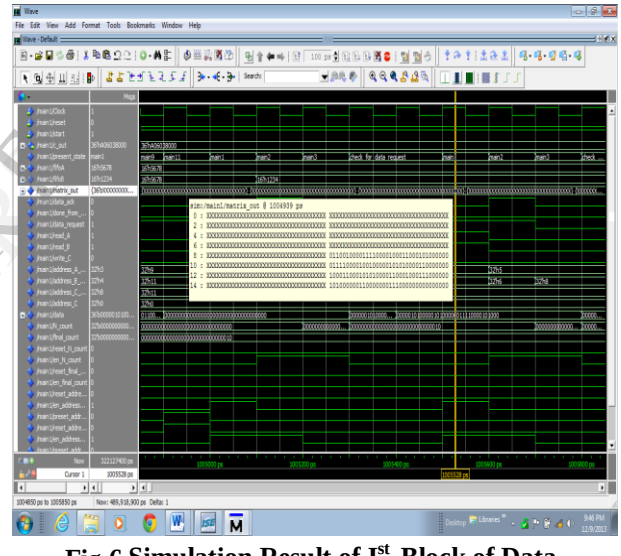

Fig-6 Simulation Result of Ist Block of Data

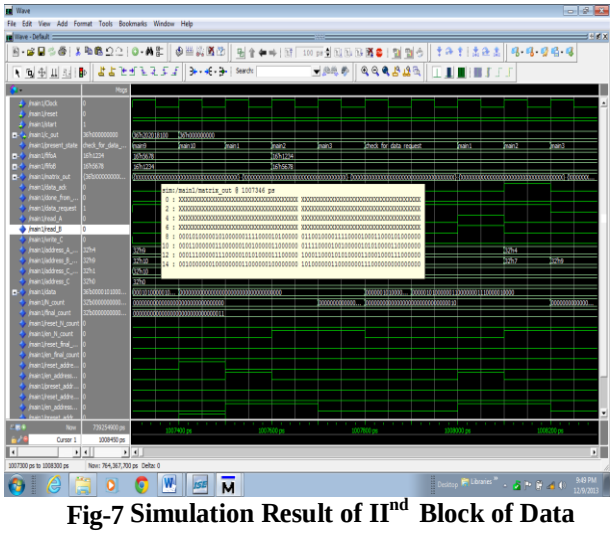

Fig-7 Simulation Result of IInd Block of Data

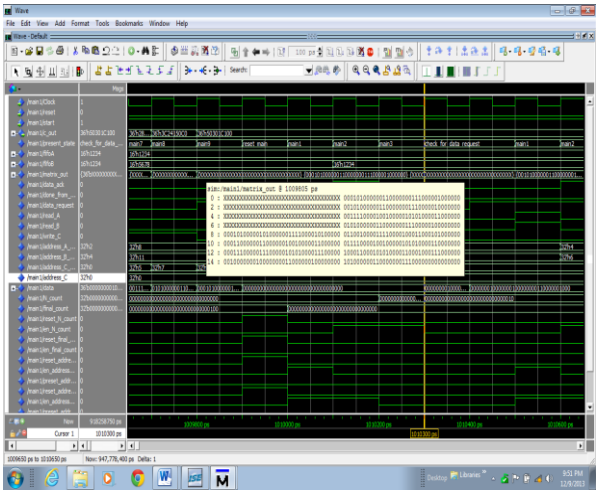

Fig-8 Simulation Result of IIIrd Block of Data

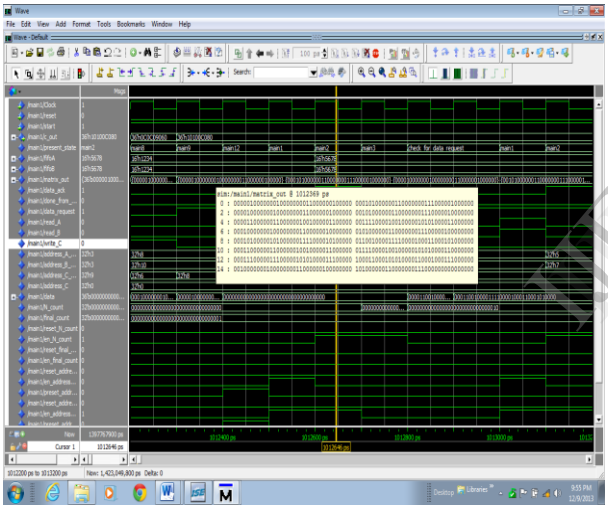

Fig-9 Simulation Result of IV^{rt} Block of Data

4. RTL AND TECHNOLOGY SCHEMATIC

4.1 RTL View:

RTL View is a Register Transfer Level graphical representation of Architecture design. This representation (.ngr file produced by Xilinx Synthesis Technology (XST)) is generated by the synthesis tool at earlier stages of a synthesis process when technology mapping is not yet completed. The goal of this view is to be as close as possible to the original HDL code. In the RTL view, the design is represented in terms of macro blocks, such as adders, multipliers, and registers. Standard combinatorial logic is mapped onto logic gates, such as AND, NAND, and OR.

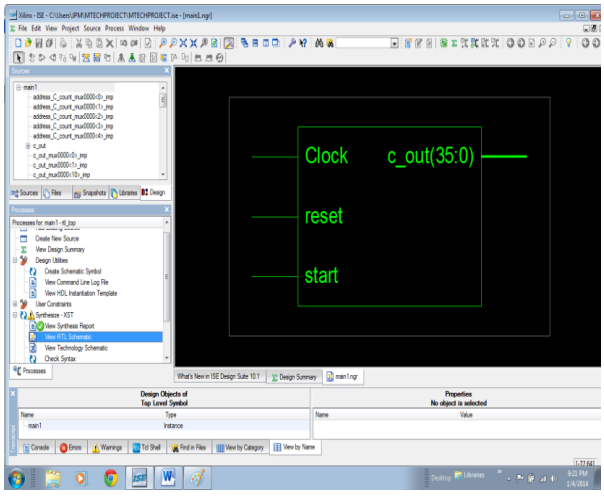

Fig-10 RTL Schematic

4.2 Technology Schematic

This schematic shows the representation of design in terms of logic elements optimized to the target Xilinx device or "technology", for example, in terms of LUTS, carry logic, IOB'S and other technology specific components. Viewing this schematic allows seeing a technology level representation of HDL optimized for a specific Xilinx architecture, which may help discover design issues early in the design process.

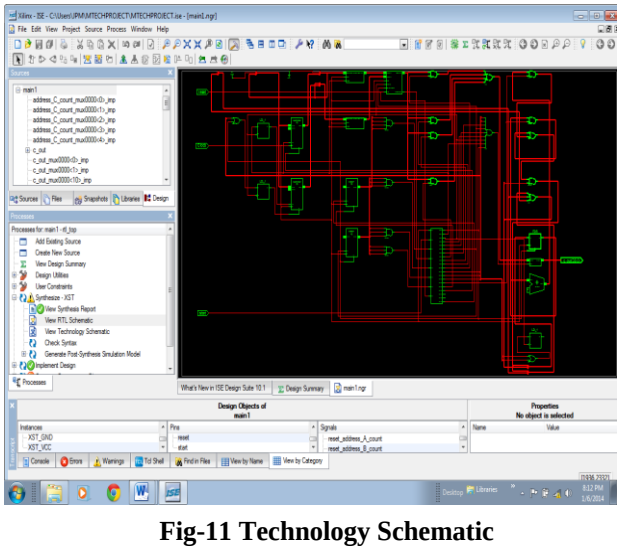

Fig-11 Technology Schematic

5. COMPARATIVE RESULTS & PERFORMANCE

The proposed architecture was modeled in VHDL hardware description language. The VHDL model was synthesized with Xilinx ISE 10.1i targeted for Spartan

3E (XC3S500e-5vq100) FPGA device from Xilinx. In order to show the performance of the proposed architecture, the same FPGA device was used. For illustration of the proposed technique, we designed 8×8 matrix multiplier for 8-bit fixed point number. The design implemented uses 4 Processing Elements. The results demonstrate that our design have fewer Utilization of FPGA resources as compared to previous Architecture [10]. Table 1 & Table 2 summarized the performance in terms of Logic Utilization, Peak Memory and Power Consumptions comparison. The main idea of the proposed Architecture is to reuse the resource and input the data in serial. In this way, the hardware cost can be dramatically decreased. There are following Comparative Results between old architecture [10] and proposed architecture.

Table-1 Hardware Cost for old Architecture

S.No	Logic Utilization	Hardware Used	% Use
1	Number of Flip Flops	365	3%
2	Number of 4 input LUTs	452	4%
3	Number of occupied Slices	293	6%
4	Total Number of 4 input LUTs	458	4%
5	Number of bonded IOBs	165	71%
6	Number of BUFGMUXs	1	14%
7	Peak Memory Usage	172 MB	
8	Total Power	0.102W	

Table-2 Hardware Cost for Proposed Architecture

S.No	Logic Utilization	Logics Used	Available Logics	% Use
1	Number of Flip Flops	176	9312	1%
2	Number of 4 input LUTs	282	9312	3%
3	Number of occupied Slices	165	4656	3%
4	Total Number of 4 input LUTs	298	9312	4%
5	Number of bonded IOBs	39	66	59%
6	Number of BUFGMUXs	2	24	8%
7	Peak Memory Usage	116 MB		
8	Total Power	0.080W		

6. CONCLUSION

Matrix multiplication can be applied extensively in many areas. Original parallel method of the implementation on Simulink Xilinx 10.1, which is implemented for on Spartan 3E in which Block consumes considerable low hardware resources, which makes it hard to be realized. We provided a new method to improve the module in order to save the resources and consume minimum power. In this way, less hardware is needed to complete the computation. Other methods can be used to optimize the matrix multiplication, such as pipeline technology.

Several design techniques such as parallel processing and pipelining are employed to achieve high performance and efficient hardware realization of the matrix multiplier using FPGA. Implementation results demonstrate the effectiveness of the proposed design technique over previous solutions. In terms of Logic Utilization, the proposed architecture is better than the implementations that are reported in the literature.

REFERENCES

[1] Yong Dou S. Vassiliadis G. K. Kuzmanov G. N. Gaydadjiev, **"**64-bit Floating-Point FPGA Matrix Multiplication", Computer Engineering, EEMCS, TU Delft, P.O. Box 5031,2600 GA delft, The Neatherlands.
[2] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes,"Computer Graphics, Principles and Practice", Addison-Wesley, second edition, 1996.
[3] P. Graham and B. Nelson, "FPGA based Sonar Processing", Proceedings of the 6th ACM / SIGDA international symposium on FPGAs, pp. 201-208, February 1998.
[4] A. Jones, A. Nayak. P. Banerjee, "Parallel Implementation of Matrix and Signal Processing Libraries on FPGAs", Proceedings of the 14th International Conference on Parallel and Distributed Computing Systems, 200. http:Hwww.ece.northwestern.edu/-kiones/papers/342-050.pdf

[5] K. Benkrid, D. Crookes, J. Smith, and A. Benkrid, 'High Level Programming for Real Time FPGA Based Video Programming', Proceedings of ICASSP'2000, Istanbul, June 2000. Volume VI, pp. 3227-3231.

[6] S.M.Qasim, S.A.Abbasi, and B.Almashary, "A proposed FPGA-bases parallel architecture for matrix multiplication", Circuits and Systems, 2008, APCCAS 2008, Pages 1763-1766, Nov 2008.

[7] T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O.Mencer, W.Luk, and P. Y. K. Cheung,"Reconfigurable computing: architectures and design methods", *IEEE Proc. Computer and Digital Techniques*,Vol. 152, No. 2, pp 193–207, Mar. 2005.

[8] L. Singhal and E. Bozorgzadeh, "Multi-layer floorplanning for reconfigurable designs,"IET: Computers and Digital Techniques, Vol.1, No. 4, pp. 276–294, July 2007.

[9] Syed M.Qasim, Ahmed A. Telba and Abdulhameed Y. AlMazroo Department of Electrical Engineering, "FPGA Design and Implementation of Matrix Multiplier Architectures for Image and Signal Processing Applications" Centre King Saud University, College of Engineering Riyadh 11421, Saudi Arabia{smanzoor, abbasi, bmashary}@ksu.edu.

[10] Xiaoxiao Jiang[1], Jun Tao[2], Department of Electrical Engineering, University of Minnesota,Twin Cities, USA,"Implementation of Effective Matrix Multiplication on FPGA", Proceeding of IEEE IC-BNMT2011.