High Speed and Area Efficient Adjoint Matrix using Booth Multiplier for FPGA Implementation

Rohit Sachan[1], Prof. Suresh. S. Gawande[2], Prof. Satyarth Tiwari[3]

M. Tech. Scholar, Department of Electronics and Communication, Rkdf college of engineering, Bhabha University, Bhopal[1]

Guide, Department of Electronics and Communication, Rkdf college of engineering, Bhabha University, Bhopal[2]

Co-guide, Department of Electronics and Communication, Rkdf college of engineering, Bhabha University, Bhopal[3]

Abstract— Due to advancement of new technology in the field of VLSI and Embedded system, there is an increasing demand of high speed and low power consumption processor. Speed of processor greatly depends on its multiplier as well as adder performance. 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. In spite of complexity involved in floating point arithmetic, its implementation is increasing day by day. Due to which high speed adder architecture become important. Several adder architecture designs have been developed to increase the efficiency of the adder. In this paper, we introduce an architecture that performs high speed IEEE 754 floating point multiplier using carry select adder (CSA). Here we are introduced two carry select based design. These designs are implementation Xilinx Vertex device family.

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

1. INTRODUCTION

With the growth in scale of integration circuits, more and more sophisticated digital signal processing circuits are being implemented in (field programmable gate array) FPGA based circuit. Indeed, FPGA have become an attractive fabric for the implementation of computationally intensive application such as digital signal processing, image, graphics card and network processing tasks used in wireless communication. These complex signal processing circuits not only demand large computational capacity but also have high energy and area requirements. Though area and speed of operation remain the major design concerns, power consumption is also emerging as a critical factor for present VLSI system designers [1]-[4]. The need for low power VLSI design has two major motivations. First, with increase in operating frequency and processing capacity per chip, large current have to be delivered and the heat generated due to large power consumption has to be dissipated by proper cooling techniques, which account for additional system cost. Secondly, the exploding market of portable electronic appliances demands for complex circuits to be powered by lightweight batteries with long times between re-charges (for instance [5].

Another major implication of excess power consumption is that it limits integrating more transistors on a single chip or on a multiple-chip module. Unless power consumption is dramatically reduced, the resulting heat will limit the feasible packing and performance of VLSI circuits and systems. From the environmental viewpoint, the smaller the power dissipation of electronic systems, the lower heat pumped into the surrounding, the lower the electricity consumed and hence, lowers the impact on global environment [6].

Matrix multiplication is commonly used in most signal processing algorithms. It is also a frequently used kernel operation in a wide variety of graphics, image processing as well as robotic applications. The matrix multiplication operation involves a large number of multiplication as well as accumulation. Multipliers have large area, longer latency and consume considerable power compared to adders. Registers, which are required to store the intermediate product values, are also major power intensive component [7]. These components pose a major challenge for designing VLSI structures for large-order matrix multipliers with optimized speed and chip-area. However, area, speed and power are usually conflicting hardware constraints such that improving upon one factor degrades the other two. The real numbers represented in binary format are known as floating point numbers. 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.

II. DIFFERENT TYPES OF ADDER

Parallel Adder:-

Parallel adder can add all bits in parallel manner i.e. simultaneously hence increased the addition speed. In this adder multiple full adders are used to add the two corresponding bits of two binary numbers and carry bit of the previous adder. It produces sum bits and carry bit for the next stage adder. In this adder multiple carry produced by multiple adders are rippled, i.e. carry bit produced from an adder works as one of the input for the adder in its succeeding stage. Hence sometimes it is also known as Ripple Carry Adder (RCA). Generalized diagram of parallel adder is shown in figure 3.

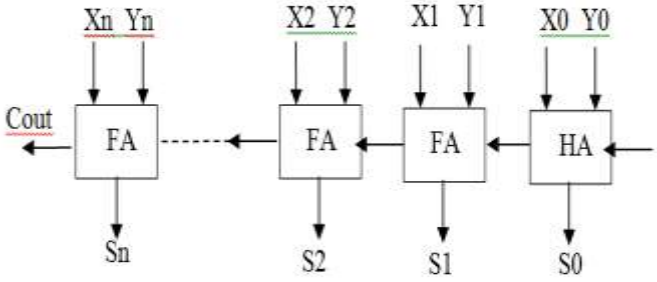

Figure 1: Parallel Adder (n=7 for SPFP and n=10 for DPFP)

An n-bit parallel adder has one half adder and n-1full adders if the last carry bit required. But in 754 multiplier's exponent adder, last carry out does not required so we can use XOR Gate instead of using the last full adder. It not only reduces the area occupied by the circuit but also reduces the delay involved in calculation. For SPFP and DPFP multiplier's exponent adder, here we Simulate 8 bit and 11 bit parallel adders respectively as show in figure 4.

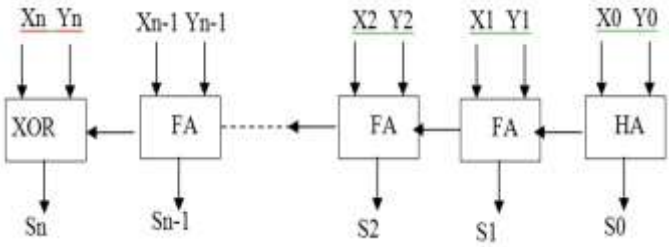

Figure 2: Modified Parallel Adder (n=7 for SPFP and n=10 for DPFP)

Carry Select Adder:-

Carry select adder uses multiplexer along with RCAs in which the carry is used as a select input to choose the correct output sum bits as well as carry bit. Due to this, it is called Carry select adder. In this adder two RCAs are used to calculate the sum bits simultaneously for the same bits assuming two different carry inputs i.e. '1' and '0'. It is the responsibility of multiplexer to choose correct output bits out of the two, once the correct carry input is known to it. Multiplexer delay is included in this adder. Generalized figure of Carry select adder is shown in figure 3.9. Adders are the basic building blocks of most of the ALUs (Arithmetic logic units) used in Digital signal processing and various other applications**.** Many types of adders are available in today's scenario and many more are developing day by day. Half adder and Full adder are the two basic types of adders. Almost all other adders are made with the different arrangements of these two basic adders only. Half adder is used to add two bits and produce sum and carry bits whereas full adder can add three bits simultaneously and produces sum and carry bits.

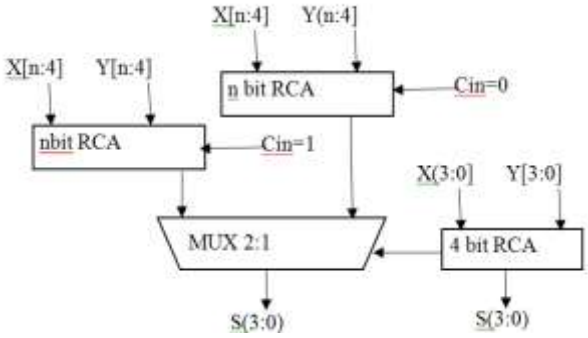

Figure 3: Carry Select Adder

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. From equation 2 we have derived for parallel computation of 3 × 3 matrix-matrix multiplication and the structure is shown in figure 4.

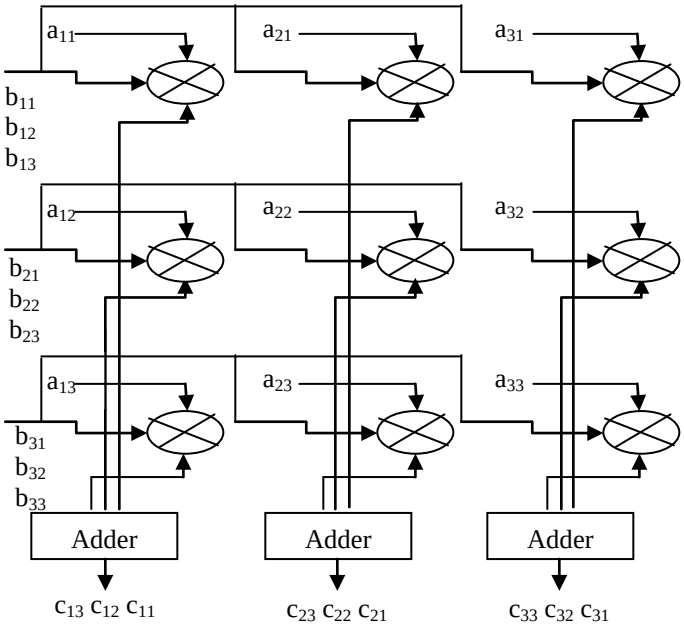

Figure 4: Proposed PPI – MO Design for n = 3

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.

Booth Multiplier

There is no need to take the sign of the number into deliberation in dealing with unsigned multiplication. However in signed multiplication the process will be changed because the signed number is in a 2's compliment pattern which would give a wrong result if multiplied by using similar process for unsigned multiplication [6]. Booth's algorithm is used for this. Booth's algorithm preserves the sign of the result. Booth multiplication allows for smaller, faster multiplication circuits through encoding the signed numbers to 2's complement, which is also a standard technique used in chip design, [6] and provides significant improvements by reducing the number of partial product to half over "long multiplication" techniques. Radix 2 is the conventional booth multiplier.

Radix 2

In booth multiplication, partial product generation is done based on recoding scheme e.g. radix 2 encoding. Bits of multiplicand (Y) are grouped from left to right and corresponding operation on multiplier (X) is done in order to generate the partial product [19]. In radix-2 booth multiplication partial product generation is done based on encoding which is as given by Table1. Parallel Recoding scheme used in radix-2 booth multiplier is shown in the Table 1.

Table 1: Booth recoding for radix 2

Q_n	Q_{n+1}	Recoded Booth	Operation
0	0	0	Shift
0	1	+1	Add x
1	0	-1	Subtract x
1	1	0	Shift

Radix-4

To further decrease the number of partial products, algorithms with higher radix value are used. In radix-4 algorithm grouping of multiplier bits is done in such a way that each group consists of 3 bits as mentioned in table 1. Similarly the next pair is the overlapping of the first pair in which MSB of the first pair will be the LSB of the second pair and other two bits. Number of groups formed is dependent on number of multiplier bits. By applying this algorithm, the number of partial product rows to be accumulated is reduced from n in radix-2 algorithm to n/2 in radix-4 algorithm. The grouping of multiplier bits for 8-bit of multiplication is shown in figure 5.

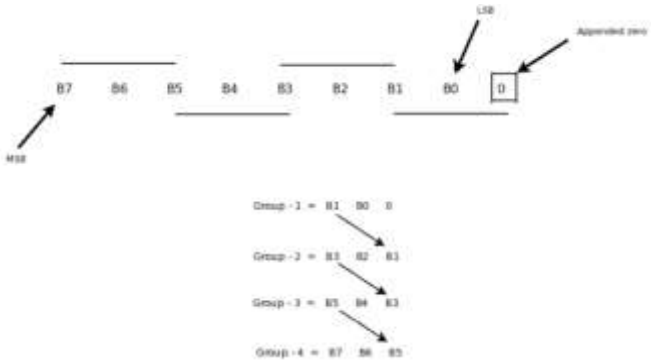

Figure 5: Grouping of multiplier bits in Radix-4 Booth algorithm

For 8-bit multiplier the number groups formed is four using radix-4 booth algorithm. Compared to radix-2 booth algorithm the number of partial products obtained in radix-4 booth algorithm is half because for 8-bit multiplier radix-2 algorithm produces eight partial products. The truth table and the respective operation is depicted in table 1. Similarly when radix-8 booth algorithm is applied to multiplier of 8-bits each group will consists of four bits and the number of groups formed is 3. For 8x8 multiplications, radix-4 uses four stages to compute the final product and radix-8 booth algorithm uses three stages to compute the product. In this thesis, radix-4 booth algorithm is used for 8x8 multiplication because number components used in radix-4 encoding style.

Table 2: Truth Table for Radix-4 Booth algorithm

B_{i+1}	B_i	B_{i-1}	Operation	Y_{i+1}	Y_i	Y_{i-1}
0	0	0	+0	0	0	0
0	0	1	+A	0	1	0
0	1	0	+A	0	1	0
0	1	1	+2A	0	0	1
1	0	0	-2A	1	0	1
1	0	1	-A	1	1	0
1	1	0	-A	1	1	0
1	1	1	-0	1	0	0

IV. SIMULATION RESULT

All the designing and experiment regarding algorithm that we have mentioned in this paper is being developed on Xilinx 6.2i 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 6.2i 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.

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

Structure	Dimension	Slice	LUTs	IOBs	Delay (ns)
Previous Design [1]	3×3	112	164	81	15.517
MM using PPI-SO		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	248	412	96	17.227
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

Most of the digital signal processing (DSP) algorithms is formulated as matrix-matrix multiplication, matrix-vector multiplication and vector-vector (Inner-product and outer-product) form. Few such algorithms are digital filtering, sinusoidal transforms, wavelet transform etc. The size of matrix multiplication or inner-product computation is usually large for various practical applications. On the other hand, most of these algorithms are currently implemented in hardware to meet the temporal requirement of real-time application [9]. When large size matrix multiplication or inner product computation is implemented in hardware, the design is resource intensive. It consumes large amount of chip area and power. With such a vast application domain, new designs are required to cater to the constraints of chip area and power and high speed.

IEEE754 standardize two basic formats for representing floating point numbers namely, single precision floating point and double precision floating point. Floating point arithmetic has vast applications in many areas like robotics and DSP. Delay provided and area required by hardware are the two key factors which are need to be consider Here we present single precision floating point multiplier by using two different adders namely modified CSA with dual RCA and modified CSA with RCA and BEC.

REFRENCES

[1] Di Yan, Wei-Xing Wang, Lei Zuo, *Member, IEEE* and Xiao-Wei Zhang, "Revisiting the Adjoint Matrix for FPGA Calculating the Triangular Matrix Inversion", IEEE Transactions on Circuits and Systems II: Express Briefs, 2020.

[2] X.-W. Zhang, L. Zuo, M. Li and J.-X. Guo, "High-throughput FPGA implementation of Matrix inversion for control systems," Accepted by *IEEE Trans. Ind. Electron.*, 2020.

[3] C. Zhang, et al,. "On the low-complexity, hardware-friendly tridiagonal matrix inversion for correlated massive MIMO systems," *IEEE Trans. Vehic. Tech.*, vol. 68, no. 7, pp. 6272-6285, Jul. 2019.

[4] Y.-W. Xu, Y. Xi, J. Lan and T.-F. Jiang, "An improved predictive controller on the FPGA by hardware matrix inversion," *IEEE Trans. Ind. Electron.*, vol. 65, no. 9, pp. 7395–7405, Sep. 2018.

[5] 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).

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] Jaenicke and W. Luk, "Parameterized Floating-Point Arithmetic on FPGAs", Proc. of IEEE ICASSP, vol. 2, (2001), pp. 897-900.

[16] Lee and N. Burgess, "Parameterisable Floating-point Operations on FPGA", Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems, and Computers, (2002).

[17] 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.