# VLSI Architecture for PPI-MO based Matrix Multiplication using Floating Point Multiplier

Shilpa Shukla[1], Prof. Amrita Khera[2]

[1]*Research Scholar, Department of Electronics and Communication, Trinity Institute of Technology & Research, Bhopal*

[2]*Assistant Professor, Department of Electronics and Communication, Trinity Institute of Technology & Research, Bhopal*

***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. 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), Binary to Execess-1 Converter.***

## I. INTRODUCTION

***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. For a number is said to be a normalized number, it must consist of'one' in the MSB of the significand and exponent is greater than zero and smaller than 1023. The real number is represented by equations (i) & (2).***

$$Z = (-1^{s}) \times 2^{(E-Bias)} \times (1.M) \quad (1)$$

$$Value = (-1^{signbit}) \times 2^{(Exponent-1023)} \times (1.Mantissa) \quad (2)$$

Biasing makes the values of exponents within an unsigned range suitable for high speed comparison.

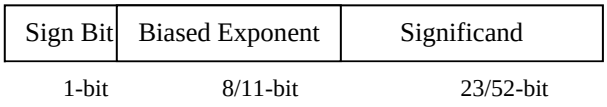


**Figure 1: IEEE 754 Single Precision and Double Precision Floating Point Format**

IEEE 754 Standard Floating Point Multiplication Algorithm

A brief overview of floating point multiplication has been explained below [5-6].

- Both sign bits $S_1$, $S_2$ are need to be Xoring together, then the result will be sign bit of the final product.
- Both the exponent bits $E_1$, $E_2$ are added together and then subtract bias value from it. So, we get exponent field of the final product.
- Significand bits $Sig_1$ and $Sig_2$ of both the operands are multiply including their hidden bits.
- Normalize the product found in step 3 and change the exponent accordingly. After normalization, the leading "1 "will become the hidden bit.

Above algorithm of multiplication algorithm is shown in Figure 2.

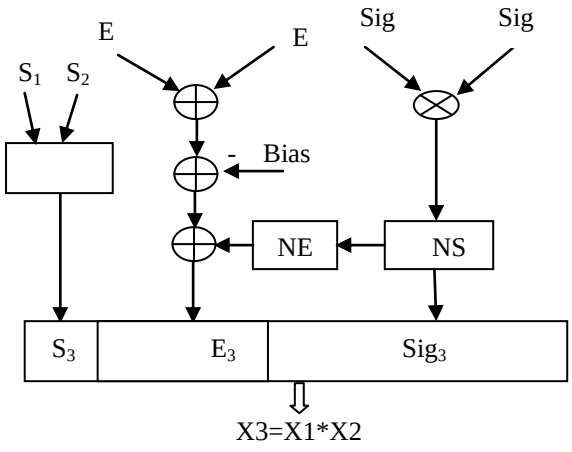


**Figure 2: IEEE754 SP FP and DP FP Multiplier Structure, NE: Normalized exponent, NS: Normalized Significand**

## 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.

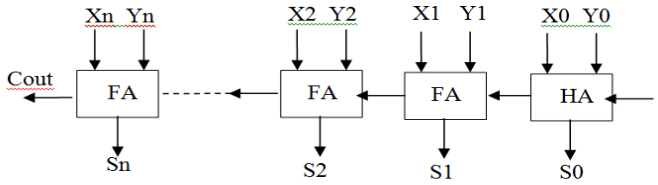


**Figure 3: Parallel Adder (n=7 for SPFP and n=10 for DPFP)**

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.

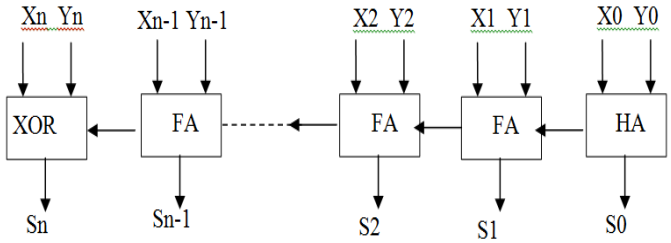


**Figure 4: Modified Parallel Adder (n=7 for SPFP and n=10 for DPFP)**

**Carry Skip Adder:-**

This adder gives the advantage of less delay over Ripple carry adder. It uses the logic of carry skip, i.e. any desired carry can skip any number of adder stages. Here carry skip logic circuitry uses two gates namely "and gate" and "or gate". Due to this fact that carry need not to ripple through each stage. It gives improved delay parameter. It is also known as Carry bypass adder. Generalized figure of Carry Skip Adder is shown in figure 5.

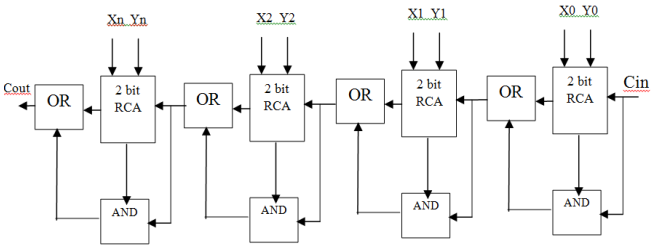


**Figure 5: Carry Skip Adder**

**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.

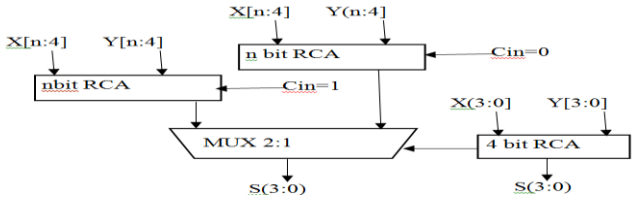


**Figure 6: Carry Select Adder**

**KS Adder: -** Processing of the KSA adder is structured with pre-processing stage, carry generator and post processing stage. Each block is having individual responsibilities. First block of KSA is Pre- Processing that will generate and propagate the carry. Processing of carry will be done over the carry processing area and all the carry signal go through the post processing block. In the pre pre-processing stage we find the, generate and propagate signals from each inputs.

Eventually, all the designing levels of digital system or IC's Packages depend on number of gates in a single chip that is also called bottom up approach. Modified KS adder can be reduced regarding the area or number of gates.

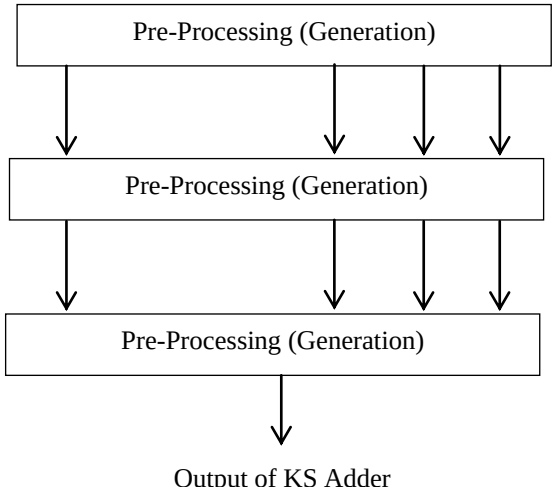


**Figure 7: Block Diagram of Kogge Stone Adder**

## III. PROPOSED DESIGN

In this design we have reduced the resource utilization in terms of number of multipliers and registers in lieu of the completion time. This design is particularly useful where resources are limited and design can be compromised on basis of increased completion time. The basic working model for a 3 × 3 matrix-matrix multiplication is shown in figure 7 below.

Considering the matrix – matrix multiplication of two n×n matrices, the calculation is performed using n number of multipliers, n number of registers and n-1 number of adders. $n^2$ cycles are required to perform the matrix multiplication operation. Each multiplier has two input ports: one each from matrix A and B. In each cycle, n numbers of multiplications are performed and the products are fed to the adder block to give a single element of the output matrix, C. The data flow to the multipliers are such that, $k^{th}$ multiplier is fed from $k^{th}$ column of matrix A and $k^{th}$ row of matrix B, where 1 < k < n. At the $k^{th}$ multiplier, each element from matrix A is repeated for n consecutive cycles whereas the elements from matrix B are cycled back after n cycles. The partial products are then fed to the adder which computes the final result.

For a better understanding of the process, let us consider the matrix multiplication for n = 3 (as shown in figure 1). In this case, 3 multipliers and 3 registers are used to calculate and store the partial products respectively. These partial products are then fed to the adder block to compute the final result. The first multiplier receives input from the first column of matrix A ($a_{k1}$) and first row of matrix B ($b_{1k}$), where. Each element of the matrix A at the first multiplier is repeated for 3 cycles, such that the data flow can be represented as $a_{11}a_{11}a_{11}\ a_{21}a_{21}a_{21}\ a_{31}a_{31}a_{31}$ .Similarly, at the first multiplier, the elements of B are repeated after 3 cycles, such that the input data-flow will be $b_{11}b_{12}b_{13}\ b_{11}b_{12}b_{13}\ b_{11}b_{12}b_{13}$.

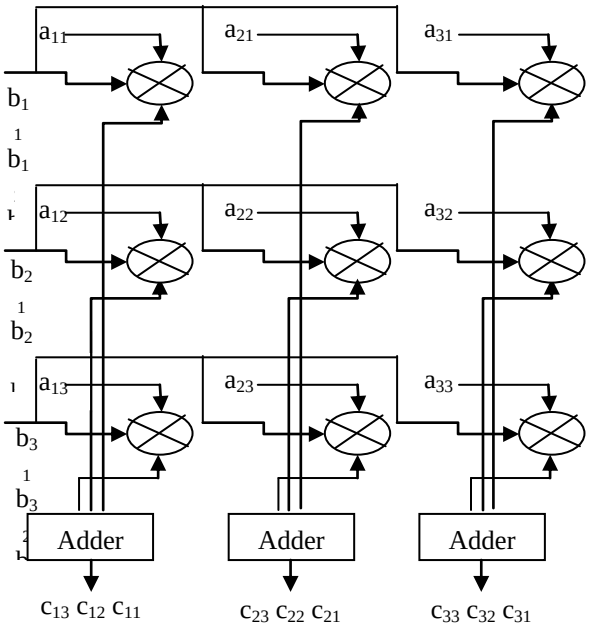


**Figure 8: Proposed PPI – MO Design for n = 3**

The other two multipliers receive the component of A and B in the similar order as the first multiplier. After the multiplication, the partial products are fed to the adder which computes the elements of output matrix C in row major order given by $c_{11}c_{12}c_{13}\ c_{21}c_{22}c_{23}\ c_{31}c_{32}c_{33}$. So the entire matrix multiplication operation is performed in $n^2$=9 cycles.

**IEEE 754 Floating Point:-** In IEEE754 standard floating point representation, 8 bit Exponent field in single precision floating point (SP FP) representation and 11 bit in double precision floating point (DP FP) representation are need to add with another 8 bit exponent and 11 bit exponent respectively, in order to multiply floating point numbers represented in IEEE 754 standard as explained earlier. Ragini et al. [10] has used parallel adder for adding exponent bits in floating point multiplication algorithm. We proposed the use of 8-bit modified CSA with dual RCA and 8-bit modified CSA with RCA and BEC for adding the exponent bits. We have found the improved area of 8-bit modified Carry select adder with RCA and BEC over the 8-bit modified CSA with dual RCA.

- Sign bit calculation

To calculate the sign bit of the resultant product for SP FP and DP FP multiplier, the same strategy will work. We just need to XOR together the sign bits of both the operands. If the resultant bit is '1', then the final product will be a negative number. If the resultant bit is '0', then the final product will be a positive number.

- Exponent bit calculation

Add the exponent bits of both the operands together, and then the bias value (127 for SPFP and 1023 for DPFP) is subtracted from the result of addition. This result may not be the exponent bits of the final product. After the significand multiplication, normalization has to be done for it. According to the normalized value, exponents need to be adjusted. The adjusted exponent will be the exponent bits of the final product.

- Significand bit calculation

Significand bits including the one hidden bit are need to be multiply, but the problem is the length of the operands. Number of bits of the operand will become 24 bits in case of SP FP representation and it will be 53 bits in case of DP FP representation, which will result the 48 bits and 106 bits product value respectively. In this paper we use the technique of break up the operands into different groups then multiply them. We get many product terms, add them together carefully by shifting them according to which part of one operand is multiplied by which part of the other operand. We have decomposed the significand bits of both the operands ain four groups. Multiply each group of one operand by each group of second operand. We get 16 product terms. Then we add all of them together very carefully by shifting the term to the left according to which groups of the operands are involved in the product term.

**Partition Multiplier:-**

Algorithm for partition method

```
t1 : in  STD_LOGIC_VECTOR (7 downto 0);
t2 : in  STD_LOGIC_VECTOR (7 downto 0);
t3 : out  STD_LOGIC_VECTOR (15 downto 0));
h1<=t1(3 downto 0);
h2<=t1(7 downto 4);
h3<=t2(3 downto 0);
h4<=t2(7 downto 4);
su1<=h1*h3;
su2<=h1*h4;
su3<=h2*h3;
su4<=h2*h4;
ad1<=("00000000" & su1);
ad2<=("0000" & su2 & "0000");
ad3<=("0000" & su3 & "0000");
ad4<=(su4 & "00000000");
t3<=ad1 + ad2 + ad3 + ad4;
```

## 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 14.1i 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 I: Comparison Result**

| Structure | Dimension | Slice | LUTs | IOBs | Delay (ns) |
|---|---|---|---|---|---|
| Previous Design [1] | 3×3 | 112 | 164 | 81 | 15.517 |
| MM using PPI-MO | | 93 | 154 | 74 | 15.058 |
| Previous Design [1] | 4×4 | 248 | 412 | 96 | 17.227 |
| MM using PPI-MO | | 221 | 388 | 92 | 15.058 |

## V. CONCLUSION

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. Among all two adders, modified CSA with RCA and BEC is the least amount of Maximum combinational path delay (MCDP). Also, it takes least number of slices i.e. occupy least area among all two adders.

## REFERENCES

[1] 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).

[2] Chiou-Yng Lee, Pramod Kumar Meher, Chia-Chen Fan, and Shyan-Ming Yuan, “Low-Complexity Digit-Serial Multiplier Over G F(2m) Based on Efficient Toeplitz Block Toeplitz Matrix–Vector Product Decomposition”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2016.

[3] K. Deergha Rao, Ch. Gangadhar and Praveen K Korrai, "FPGA Implementation of Complex Multiplier Using Minimum Delay Vedic Real Multiplier Architecture", IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON) Indian Institute of Technology (Banaras Hindu University) Varanasi, India, Dec 9-11, 2016.

[4] Ms. S. V. Mogre and Mr. D. G. Bhalke "Implementation of High Speed Matrix Multiplier using Vedic Mathematics on FPGA", 2015 International Conference on Computing, Communication Control and Automation.

[5] Pramod Kumar Meher, "Hardware-Efficient Systolization of DA-Based Calculation of Finite Digital Convolution," IEEE Transaction on Circuits and Systems, vol. 53, no. 8, pp. 707 - 711, 2006

[6] Ju-Wook Jang, Seonil B. Choi, and Viktor K. Prasanna, "Energy- and Time-Efficient Matrix Multiplication on FPGAs", IEEE Transaction on Very Large Scale Integration (VLSI) Systems, vol. 13, no. 11, pp. 1305 – 1319, 2005

[7] S. Tugsinavisut, S. Jirayucharoensak and P. A. Beerelt, "An Asynchronous Pipeline Comparisons with Applications to DCT Matrix-vector Multiplication," in Proceedings of the 2003 International Symposium on Circuits and Systems(ISCAS), vol. 5, pp. V-361 - V-364, 2003.

[8] Amira, A. Bouridane, and P. Milligan, "Accelerating matrix product on reconfigurable hardware for signal processing," in Proceedings 11th International Conference on Field-Programmable Logic and Its Applications (FPL), pp. 101 – 111, 2001.

[9] O. Mencer, M. Morf, and M. J. Flynn, "PAM-Blox: High performance FPGA design for adaptive computing," in Field Programmable Custom Computing Machines (FCCM), pp. 167 – 174, 1998.

[10] Pramod Kumar Meher, "Hardware-Efficient Systemization of DA-Based Calculation of Finite Digital Convolution," IEEE Transaction on Circuits and Systems, vol. 53, no. 8, pp. 707 - 711, 2006

[11] Ju-Wook Jang, Seonil B. Choi, and Viktor K. Prasanna," Energy- and Time-Efficient Matrix Multiplication on FPGAs", IEEE Transaction on Very Large Scale Integration (VLSI) Systems, vol. 13, no. 11, pp. 1305 – 1319, 2005.

[12] Campbell, Scott J. and Sunil P.Khatri. "Resource and delay efficient matrix multiplication using newer FPGA devices" ACM Great Lakes Symposium on VLSI (2006).

[13] C Paidimarri.A, A.Cervero, P.Brisk and P.Ienne. "FPGA Implementation of a single – precision floating-point multiply-accumulator with singlie-cycle accumulation," proceedings of the IEEE symposium on Field Programmable Custom Computing Machines, April 5-7,2009 Karchi, Pakistan, pp-267-270.