Implementation and Analysis of Modified Double Precision Interval Arithmetic Array Multiplication

Krutika Ranjan Kumar Bhagwat[#1], Prof. Tejas V. Shah [*2], Prof. Deepali H. Shah [#3]

[#] *Instrumentation & Control Engineering Department,*

[*]*Instrumentation & Control Engineering Department, Gujarat Technological University*

L. D. College of Engineering, Ahmedabad-380015, Gujarat, India

[*]*S.S College of Engineering, Bhavnagar - 364060, Gujarat, India*

Abstract— This paper presents the design of a 64 bit array multiplier that performs interval multiplication. This multiplier requires carry save adders instead of full adders that reduces the delay in respect of conventional array multiplier. The 64 bit multiplication requires 53 x 53 multiplication which is done by array multiplier it has n*(n-1) CSA, where n=53 so, n*(n-1) = 53 *52= 2756 CSA is used. Arrangement of 2756 CSA is used to add partial products of multiplier. This multiplier is based on interval arithmetic which provides the better accuracy, by removing rounding off error over conventional floating point multiplier. There is performance improvement over software implementation of interval arithmetic, but it requires slightly more area rather than conventional floating point unit.

Keywords- Double Precision, Interval Multiplication, Significand multiplier , Array Multiplier.

I. DOUBLE PRECISION

IEEE 754 standard defines double precision as 1 sign bit , 11 bits for exponent ,53 bits for (52 explicitly stored) significand precision .

The format is written with the significand having an implicit integer bit of value 1, unless the written exponent is all Zeros. With the 52 bits of the fraction significand appearing in the memory format, the Total precision is therefore 53 bits (approximately 16 decimal digits, $53 \log_{10}(2) \approx 15.955$) . [4]

II. INTERVAL MULTIPLICATION

Multiplication of the intervals $x = [x_l , x_u]$ and $y = [y_l , y_u]$ is defined as:

$Z = x * y$

$= [\min(x_l y_l, x_l y_u, x_u y_l, x_u y_u), \max(x_l y_l, x_l y_u, x_u y_l, x_u y_u)]$

The interval multiplier shown in figure 2 has input and output registers, sign logic, an exponent adder and a significand multiplier with rounding and normalization logic. The input and output registers are each 64 bits and two multiplexer with control signal t_x ,t_y are used .

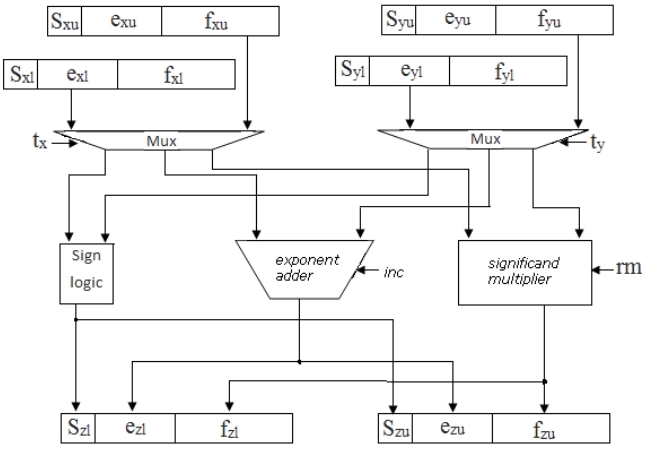

Fig. 1 Interval multiplier

The sign logic computes the sign of the result by performing the exclusive-or of the sign bits of the input operands. The exponent adder performs an 11-bit addition of the two exponents and subtracts the exponent bias of 1023. The significand multiplier performs a 53-bit by 53-bit array multiplication. If the most signicant bit of the product is one, the normalization logic shifts the product right one bit and increments the exponent. The rounding logic rounds the product to 53 bits based on a rounding mode (r_m) is round to nearest even. [10]

III. SIGNIFICAND MULTIPLIER(ARRAY MULTIPLIER USING CSA)

m x n bit multiplication can be viewed as forming n partial products of m bits each, and then summing the appropriately shifted partial products to produce an m+n bit result p. Therefore, generating partial products consist of the logical Anding of the appropriate bits of the multiplier and multiplicand. Each column of partial products must then be added and, if necessary, any carry values passed to the Next column. Simple array multiplication using full adder is shown in figure 2. [4]

Partial products are added using carry save adder instead of full adder which reduces delay. Full adder is replaced with CSA given in figure 3. The idea is

to take 3 numbers that we want to add together, x + y + z, and convert it into 2 numbers c + s such that x + y + z = c + s. In carry save addition, we refrain from directly passing on the carry information until the very last step. [13]

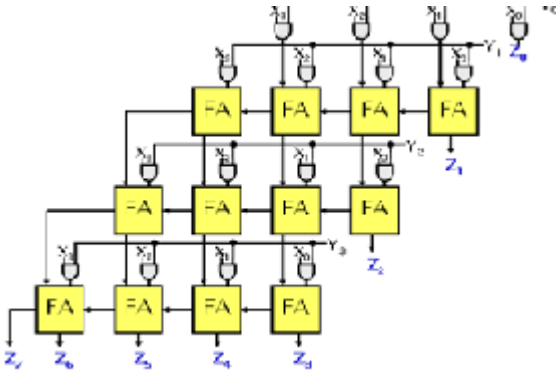

Figure 2: simple array multiplication

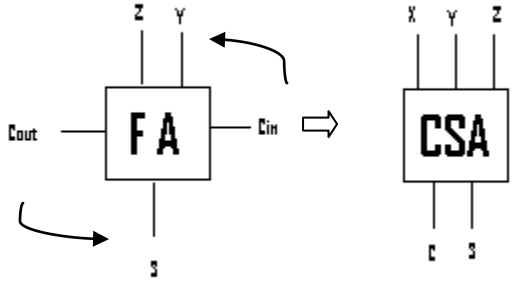

Figure 3: Full adder is replaced with CSA

The significand multiplier performs a 53-bit by 53-bit multiplication. If the most significant bit of the product is one, the normalization logic shifts the product right one bit and increments the exponent. Arrangement of CSA for
p [22:0] is shown in figure 5.

The inputs are a [52:0] and b [52:0] to generate product p [105:0]. Consider j as carry save adder and s as sum of CSA and c as carry of CSA and aobo as partial product of a[0] and b[0], and Kxy as partial product of a[x] and b[y] in modify array multiplication for half precision as reference to double precision arrangement shown in figure 4 which has product p[22:0] when the inputs are a[10:0] and b[10:0].

Arrangement of CSA for half precision the significand multiplier performs an 11-bit by 11-bit multiplication is given in figure4. If the most signicant bit of the product is one, the normalization logic shifts the product right one bit and increments the exponent.

N-bit unsigned array multiplier required

n*(n-1) CSA= 53 *52 = 2756.

IV. IMPLEMENTATION

The signs of the endpoints of the intervals x and y indicate whether x and y are greater than Zero, less than Zero, or contain Zero. This results in nine possible cases, as shown in Table1 .

All 9 cases for interval multiplication for lower
Interval Z_l and upper interval Z_u are given in Table 1, when both x and y contain Zero,

$$mn = \min(\nabla xlyu, \nabla xuyl) \text{ and}$$
$$mx = \max(\Delta xlyl, \Delta xuyu).$$

Take e_{xl} as $(10101010101)_b = (555)_h$ and e_{xu} as $(11001100110)_b = (666)_h$ and e_{yl} as $(11100011100)_b = (71c)_h$ and e_{yu} as $(11110000111)_b = (787)_h$ and f_{xl} as $(15555555555555)_h$ and fx_u as $(19999999999999)_h$ and f_{yl} as $(1c71c71c71c71c)_h$,f_{yu} as $(1e1e1e1e1e1e1e)_h$.

For case 1 Inputs are $S_{xl}=0$, $S_{xu}=0$,$S_{yl}=0$, $S_{yu}=0$ that generates outputs are $S_{zl} = \text{xor}(S_{xl}, S_{yl}) = 0$ and output $S_{zu}= \text{xor}(S_{xu},S_{yu})= 0$. For exponent add internal wire e_{zl} is the addition of ex_l and e_{yl}. That generates output $e_{zl} = (10001110001)_b$ and overflow_flag1 (o_f1) = 1. For bias 1023 output $e_{zl}= e_{zl}-1023 = (00001110010)_b$ and flag1 =0. For exponent add Internal wire e_{zu} = exponent add(e_{xu}, e_{yu}) = $(10111101101)_b$ and overflow_flag2(o_f2) = 1. For bias 1023 output $e_{zu} = e_{zu}-1023 = 00111101110$ and flag1=0.And outputs are $f_{zl} = (f_{xl}*f_{yl})$ is equals to $(ecf684bda12f684c)_h$, $f_{zu} = (f_{xu} * f_{yu})$ is equals to $(2edededededede)_h$.

In figure 5 and 6 case1 simulation reports are given. Same calculation up to cases for 64 bit interval arithmetic array multiplication is given in Table 1.

For case nine inputs are $S_{xl}=1$,$S_{xu}=0$,$S_{yl}=1$, $S_{yu}=0$ generate outputs are $S_{zl}= \text{xor}(S_{xl}, S_{yu})=1$, $S_{zu}= \text{xor}(S_{xl}, S_{yl})= 0$,$e_{xl}$ = 3'h= 555 ,e_{xu} = 3'h= 666,e_{yl} = 3'h=71c ,e_{yu}=3'h= 787.

For case nine consider four different conditions

1. $f_{xl} > f_{xu}$ and $f_{yl} > f_{yu}$,
2. $f_{xl} > f_{xu}$ and $f_{yl} < f_{yu}$,
3. $f_{xl} < f_{xu}$ and $f_{yl} < f_{yu}$,
4. $f_{xl} < f_{xu}$ and $f_{yl} > f_{yu}$

VERILOG HDL is used for programming by Xilinx ISE Design Suite 13.2 for synthesis and schematic and simulation is done here with the help of ISIM 13.2.

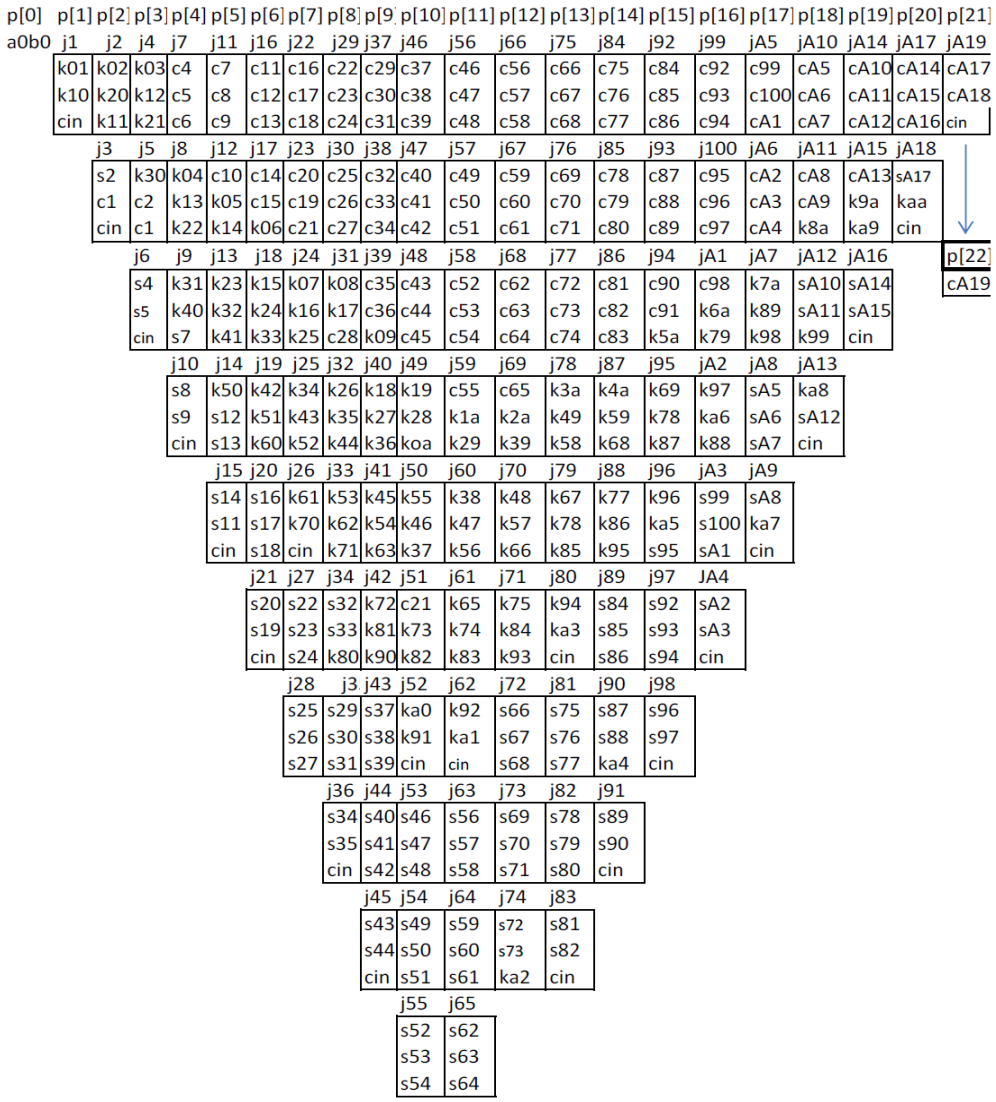

Figure 4. Modify array multiplication for half precision as reference to double precision arrangement

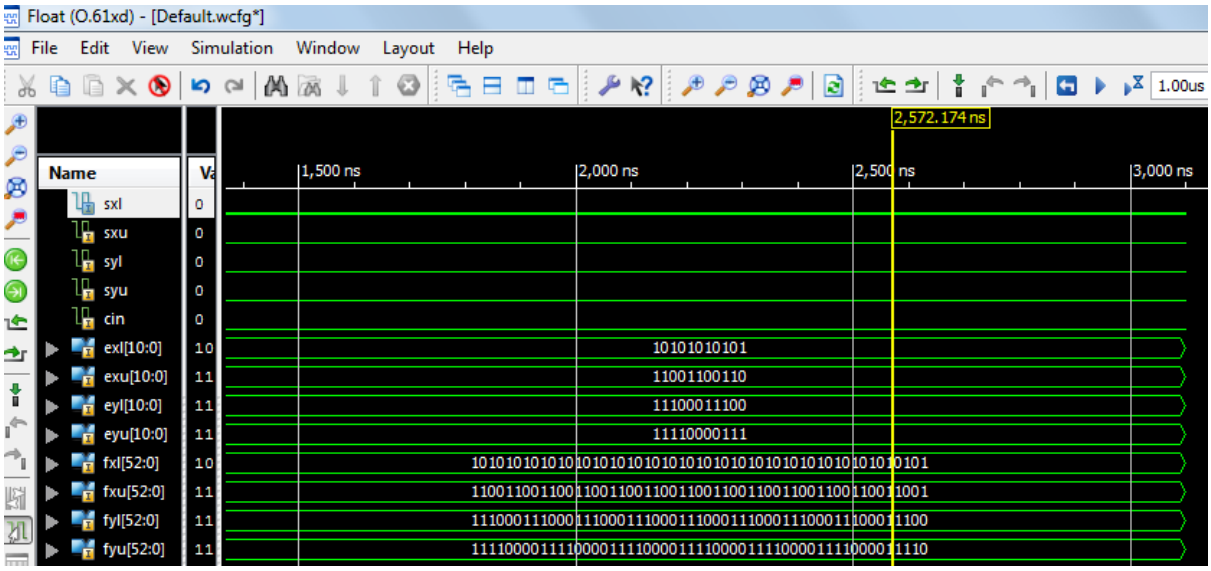

Fig.5 case1 inputs waveforms

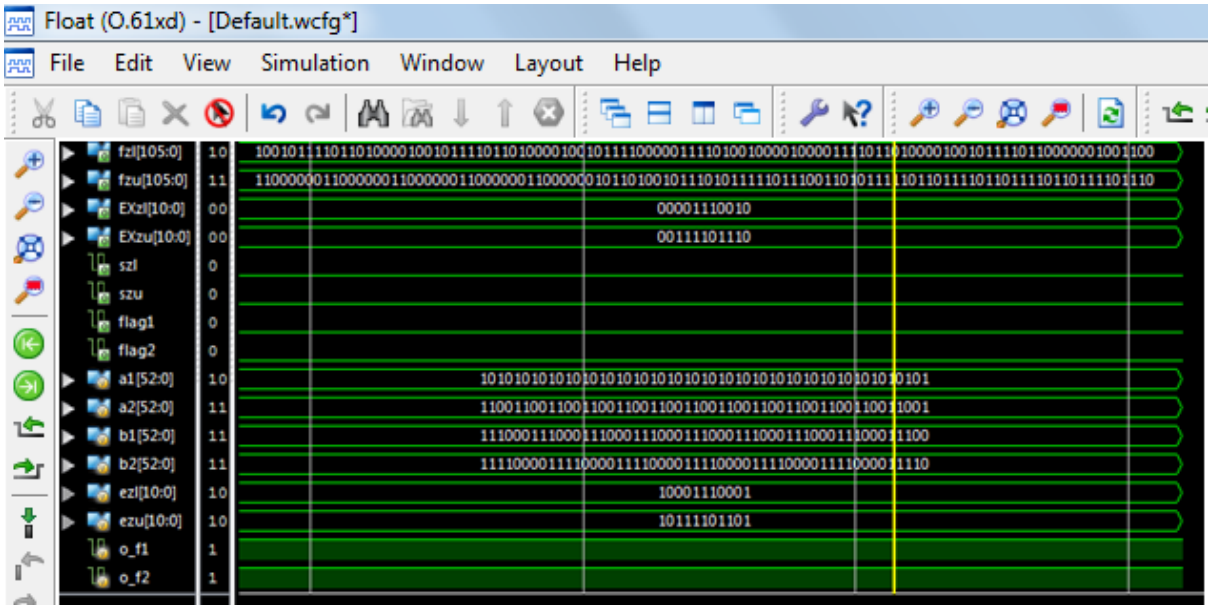

Fig.6 case1 outputs waveforms

TABLE I
CASES FOR 64 BIT INTERVAL ARITHMETIC ARRAY MULTIPLICATION

		INPUT		o/p	Z	Ex ADD		Bias 1023		
Case	Condition	S_{xl}	S_{yl}	S_{zl}	$Z_l = X_l Y_l$	Internal wire ez_l	OF	$E_{zl} = e_{zl}$-1023	f	f_{Zl} (16'h)
1	$X_l > 0, Y_l > 0$	0	0	0		10001110001	1	1110010	0	ECF684BDA12F684C
		Sx_u	Sy_u	SZ_u	$Zu = XuYu$	Internal wire ez_u	OF	$E_{zu} = eZ_u$-1023	f	f_{Zu} (16'h)
		0	0	0		10111101101	1	111101110	0	2EDEDEDEDEDEEE
2	$X_l > 0, Y_u < 0$	0	1	1	Zl = XuYl	10110000010	1	110000011	0	82BBBBBBBBBBBBC
		0	1	1	Zu=XlYu	10011011100	1	1101110	0	8275F5F5F5F5F5F6
3	$X_u < 0, Y_l > 0$	1	0	1	Zl=Xlyu	10011011100	1	11011101	0	8275F5F5F5F5F5F6
		1	0	1	Zu=XuYl	10110000010	1	110000011	0	82BBBBBBBBBBBBC
4	$X_u < 0, Y_u < 0$	1	1	0	Zl=XuYu	10111101101	1	111101110	0	2EDEDEDEDEDEEE
		1	1	0	Zu=XlYl	10001110001	1	1110010	0	ECF684BDA12F684C
5	$X_l < 0 < X_u, Y_l > 0$	1	0	1	Zl=XlYu	10011011100	1	110000011	0	8275F5F5F5F5F5F6
		0	0	0	Zu=XuYu	10111101101	1	111101110	0	2EDEDEDEDEDEEE
6	$X_l < 0 < X_u, Y_l < 0$	1	1	1	Zl=XuYl	10110000010	1	110000011	0	82BBBBBBBBBBBBC
		0	1	0	Zu=XlYl	10001110001	1	1110010	0	ECF684BDA12F684C
7	$X_l > 0, Y_l < 0 < Y_u$	0	1	1	Zl=XuYl	10110000010	1	110000011	0	82BBBBBBBBBBBBC
		0	0	0	Zu=XuYu	10111101101	1	111101110	0	2EDEDEDEDEDEEE
8	$X_u < 0, Y_l < 0 < Y_u$	1	1	1	Zl=XlYu	10011011100	1	110000011	0	8275F5F5F5F5F5F6
		1	0	0	Zu=XlYl	10001110001	1	1110010	0	ECF684BDA12F684C
9	$X_l < 0 < X_u, Y_l < 0 < Y_u$	1	1	1	Zl=Xuyu	10111101101	1	111101110	0	2EDEDEDEDEDEEE
		0	0	0	Zu=XlYu	10111101101	0	110000011	0	8275F5F5F5F5F5F6

TABLE II
ANALYSIS REPORT

Device utilization summary for 64 bit multiplication		
Area analysis	floating point	interval arithmetic
Number of Slices	4212	8603
Number of Slice Flip Flops		234
Number of 4 input LUTs	7326	14967
ımber of Ios	261	499
Number of bonded IOBs	261	499
IOB Flip Flops		2
Number of GCLKs	1	1
Real time delay	220 ns	358 ns
Maximum combinational path delay analysis in ns		
Critical path delay	465.005	421.563
Memory in kilobytes		
Total memory usage	287584	334140

V. CONCLUSION

Interval arithmetic provides reliability and accuracy by computing a lower and upper bound in which result is guaranteed to reside. Concept of carry look ahead for 11 bit exponent adder is used which reduces the delay. Concept of carry save adder in array multiplication is used instead of half adders and full adders which reduces the number of gates and delay. 334140 kilobytes memory is required. 421.563 ns is the maximum critical path delay for 64 bit interval arithmetic array multiplier.

64 bit interval arithmetic array multiplier requires almost twice real time delay compared to 64 bit floating point array multiplier. So speed of interval arithmetic array multiplier decreases and area increases.

REFERENCES

[1] Josh Milthorpe and Alistair Rendell "Learning to live with errors: A fresh look at floating-point computation", Australian National University, Computing Conference 2005

[2] Gupte, ruchir "Interval arithmetic logic unit for dsp and control applications", Electrical and Computer Engineering, Raleigh 2006

[3] Rajashekar Shettar, Dr.R.M.Banakar and Dr. P.S.V.Nataraj, "Design and Implementation of Interval Arithmetic Algorithms", First International Conference on Industrial and Information Systems, ICIIS 2006, 8 - 11 August 2006, Sri Lanka

[4] Wikipedia, the free encyclopedia

[5] Rajashekar Shettar, Dr.R.M.Banakar and Dr. P.S.V.Nataraj, "Implementation of Interval Arithmetic Algorithms on FPGAS", International Conference on Computational Intelligence and Multimedia Applications 2007, © 2007 IEEE

[6] Alexandru Amaricai Mircea Vladuaiu Lucian Prodan Mihai Udrescu Oana Boncalo "Design of Addition and Multiplication Units for High Performance Interval Arithmetic Processor", Computer Science and Engineering Department, ©2007 IEEE

[7] Michael J. Schulte, Member, IEEE, and Earl E. Swart Zlander Jr. , Fellow, IEEE, "A Performance Comparison Study on Multiplier Designs" , IEEE Transaction On Computers, May 2000

[8] Yong Dou S. Vassiliadis G. K. KuZmanov G. N. Gaydadjiev , "64-bit Floating-Point FPGA Matrix Multiplication" , National Laboratory for Computer Engineering, FPGA'05, Monterey, California, USA, February 20–22, 2005

[9] Anane Nadjia, Anane Mohamed, Bessalah Hamid, Issad Mohamed & Messaoudi khadidja, "Hardware Algorithm for Variable Precision Multiplication on FPGA" © 2009 IEEE

[10] James E. Stine and Michael J. Schulte "A Combined Interval and Floating Point Multiplier", Computer Architecture and Arithmetic Laboratory, Electrical Engineering and Computer Science Department, Lehigh University, Bethlehem, PA 18015

[11] Sparc Architecture Manual

[12] Lab-Report ECAD, "4bit multiplier using Mentor Graphics"

[13] Prof. LohCS3220- Processor Design "Carry-Save Addition" - Spring 2005, February 2, 2005

[14] "Carry Save Adder Trees in Multipliers" E C E N 6 2 6 3 A d v a n c e d V L S I D e s i g n November 3, 1999

[15] C. N. Marimuthu1, P. Thangaraj "Low Power High Performance Multiplier", Anna University,Tamil nadu , India