Journal of Theoretical and Applied Information Technology
20th June 2014. Vol. 64 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

IMPLEMENTATION OF CONFIGURABLE FLOATING POINT MULTIPLIER

[1]GOKILA D, [2]Dr. MANGALAM H
[1]Department of ECE, United Institute of Technology, Coimbatore, Tamil nadu, India
[2]Department of ECE, Sri Krishna College of Technology Coimbatore ,Tamil nadu ,India
E-mail: [1]gokila_dr@yahoo.com, [2]hmangalam2@gmail.com

ABSTRACT

Floating point multiplier is widely used in digital signal processing applications. The performance of Field Programmable Gate Arrays (FPGAs) used for floating point application is low, because of complexity in operations. This creates less interest in making FPGAs for use in floating point applications. So we are going for the reconfigurable floating point multiplier which provides better utilization of the multiplier in all applications and functions. This performs double precision operation or single precision operation. The credibility of our multiplier is tested using a standard bench mark circuit namely 4 tap FIR filter. The implementation shows a better performance with respect to delay.
Keywords: *Double Precision, Single Precision, Reconfigurable, Floating Point Multiplier (FPM), FIR Filter.*

1. INTRODUCTION

High processing performance and low power dissipation are the most important objectives in many multimedia and digital signal processing (DSP) systems, where multipliers are always the basic arithmetic unit and significantly influence the system's performance and power dissipation. Multipliers using floating point numbers are in great demand because floating point numbers have good precision, since they never deliberately discard information.So a fast and energy-efficient floating point unit is always needed in electronics industry. Field Programmable Gate Arrays (FPGA's) are broadly used for scientific computation because of the ease of customizing the hardware for the application. The limited size and architecture of FPGAs are not well-suited for floating-point applications. On the other hand, ASICs can be very efficient at floating-point operations, but lack the programmability and flexibility that is desired in many situations, and the cost of an ASIC can be prohibitively high. By overcoming the limitations of FPGAs, it will be very attractive platform for floating point applications. So for the better utilization of the floating point multiplier unit, reconfigurable computing is added. A new computing method using reconfigurable architectures promises an intermediate trade-off between flexibility and performance. Reconfigurable computing uses hardware that can be adapted at run-time to facilitate greater flexibility without compromising on performance. The re-configurability of the hardware permits adaptation of the hardware for specific computations in each application to achieve higher performance compared to software. Here the re-configurability is applied to perform single precision and double precision floating point multiplication. In order to evaluate the proposed FPGA architecture, one of the most widely performed operations in DSP namely finite impulse response (FIR) filter is used for evaluation. A 4 tap FIR filter - a standard benchmark circuit is built and implemented.

The rest of this paper is organized as follows Section 2 gives an overview of related works. Section 3 & 4 briefs about the floating point representation and multiplication respectively. Section 5 describes the architecture of the Floating Point multiplier followed by the implementation and discussion in section 6 and conclusion in section7.

2. RELATED WORKS

In FPGA's, re-configurability gives significant area utilization and delay improvements. A number of works has been proposed based on the configurability. Akkas [1] has produced the multiplier which is configured to perform either one quad precision multiplication or two double precision multiplications in parallel. It takes three cycles to perform quadruple precision multiplication and can produce a quadruple precision product every other cycle. And two double precision operations in parallel will take two cycles. Diniz and Govindu[3] has presented the design of a field programmable dual precision multiplier. By getting knowledge

Journal of Theoretical and Applied Information Technology
20th June 2014. Vol. 64 No.2
© 2005 - 2014 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

from these, the work is proposed for doing one double precision multiplication or one single precision multiplication. Mainly the multiplier work is based on the Akkas design [1]. But the difference is that they have used two multipliers for lower precision multiplication. These same multipliers are used in multi cycle for higher precision multiplication. Due to this structure the delay of multiplication operation is high in previous works. In proposed design, Instead of two simple multipliers, Radix-4 booth concept and Wallace tree structured multiplier is used, so that, the speed of operation can be improved because of single cycle utilized for both single and double precision multiplication. This is the main advantage of this design. The reconfiguration can be obtained by just using control signal for multiplexers. Reconfiguration time is very low because it involves only changing the control signal for the multiplexers. But it has a disadvantage of having little more area than other works due to tree structured multiplier. And then the delay needed to perform the single precision operation is slightly high. The main advantage is flexibility in Floating Point Multiplier for FPGA architectures so that can both double precision multiplication and single precision multiplication can be performed. The speed of multiplication operation is improved using XOR based conditional select adder [10].

3. FLOATING POINT REPRESENTATION

In general, a floating point number can be represented as

$$\pm M \times B^E$$

Where M is the mantissa
E is the exponent
B is the base

For binary case, the floating point number is represented as

$$(-1)^s \times M \times 2^E$$

where 2 is representing the implied base. Based on IEEE 754 standard, floating point number consist of three fields

1) a sign bit (S)
2) biased exponent (E)
3) mantissa (M)

The IEEE 754 floating-point standard uses 32 bits to represent a single precision floating-point number, including a single sign bit, exponent bits with bit width 8 and 23 bits of mantissa. The mantissa has effectively 24 bits including 1 implied bit to the left of the decimal point not explicitly represented in the notation.

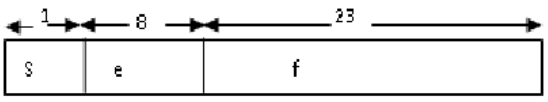


Fig 1. Ieee 754 Single Precision Floating Point Number

IEEE 754 uses 64 bits to represent double precision floating point number, including 1 sign bit, exponent bits with bit width 11 and 52 bits of mantissa. The mantissa has effectively 53 bits including 1 implied bit to the left of the decimal point not explicitly represented in the notation.

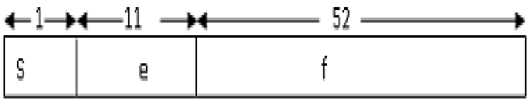


Fig 2. Ieee 754 Double Precision Floating Point Number

Bias value for the 8 bit and 11 bit exponent is 127 and 1023 respectively, then the representation is as follows

$$X = (-1)^S x 1.f x 2^{(e-127)}$$

$$X = (-1)^S x 1.f x 2^{(e-1023)}$$

Floating point numbers are having higher precision compared to fixed point numbers so that discarding of information is low.

4. FLOATING POINT MULTIPLICATION

Consider the two floating point numbers X1 = (s1, e1, f1) and X2 = (s2, e2, f2) each consists of

- Sign bit
- Exponent bits
- Mantissa bits

Then Floating point multiplication Xp can be obtained using following steps.

$$s_p = s_1 \oplus s_2$$
$$e_p = e_1 + e_2 - \text{bias}$$
$$1.f_p = 1.f_1 \times 1.f_2$$

This equation can be formed as data path as shown in Fig.3.

In mantissa adjust block the normalization operation is used, based on that the exponent value has been changed. For double precision multiplication, the mantissas are getting multiplied using 53 x 53 bit multiplier. This multiplier can be configured as 24 x 24 bit multiplier; this will help to perform single precision multiplication. This single precision multiplier will take the inputs from

LSB side of the inputs which is applied for double precision multiplication

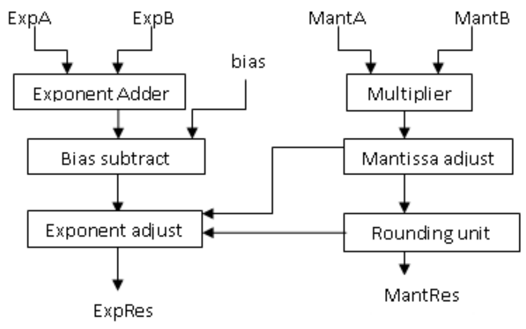


Fig 3. Data Path Of Floating Point Multiplication

In this proposed work, Radix-4 modified booth algorithm with Wallace tree structure is used to perform the mantissa multiplication. Booth encoding is used to reduce the number of partial products into half the number of bits in multiplier (X). Due to this, number of levels in Wallace structure would be reduced. Then partial products have been added using number of full adders in Wallace structure to produce the final product.

5. MULTIPLIER UNIT

The structure consists of the components for double precision multiplication. One of the inputs is given as input for booth encoder and then output from this will drive the partial product generator. Another input for partial product generator is given which is same as the second mantissa value. This will produce the partial products in 27 rows. And then these all the partial products are compressed using number of full adders and half adders to get the final sum.

In this design multiplier [10] block consists of following blocks

1) Booth Encoder

2) 53 x 53 bit Partial Product Generator

3) Wallace structure

4) XCSA adder

5.1 Booth Encoder

Parallel Multiplication using basic Booth's Recod ing algorithm technique based on the fact that partial product can be generated for group of consecutive 0's and 1's which is called as Booth's recoding. These Booth's Recoding algorithm [6] is used to generate efficient partial product. These

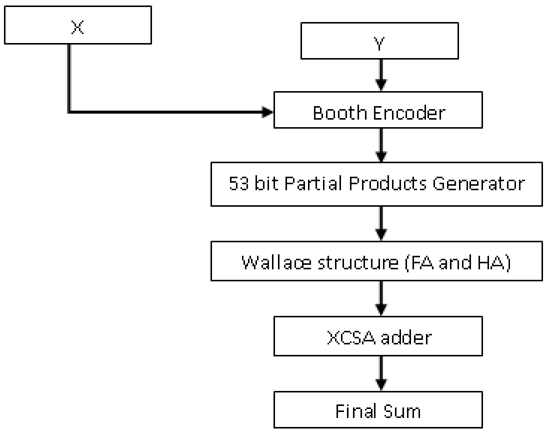

X
Y
Booth Encoder
53 bit Partial Products Generator
Wallace structure (FA and HA)
XCSA adder
Final Sum

Fig 5. Proposed Multiplier Structure

partial products always have large number of bits than the input number of bits. This partial product width usually depends upon the radix scheme used for recoding.

5.1.1Modified booth algorithm:

One of the solutions of realizing high speed multipliers is to enhance parallelism which helps to decrease the number of subsequent calculation stages. The first version of the booth algorithm [8] (radix-2) had two disadvantages. They are:

1) The number of add subtract operations and the number of shift operations become variable and becomes inconvenient in designing parallel multipliers.

2) The algorithm becomes inefficient when there are isolated 1's.

These drawbacks are overcome by using modified radix-4 booth algorithm which scans strings of three bits with the algorithm.

5.1.2 Algorithm

◊ Extend the sign bit by one position if necessary to ensure that n is even.

◊ Add a zero to the right of the LSB of the multiplier.

◊ Based on the value of each vector, each partial product will be 0, +y,-y, +2y or -2y.

The negative values of y are made by taking the two's complement. The multiplication of y is done by shifting y by one bit to the left. Thus, in any case, only n/2 partial products are generated in designing of n-bit parallel multipliers. The least significant block (LSB) uses only two bits of the multiplier, and assumes a zero for the third bit. The overlap is necessary so as to know what happened in the last block, since the MSB of the block acts like a sign bit. The modified booth algorithm using radix 4 method is the efficient technique. Based on

this the booth encoder [10] is designed with three basic operator signals.
1. Direction- Direction operator is used to choose either the normal multiplicand(X) or inverse of multiplicand (~X).
2. Shift - Shift operator is shifting the bits by one position to left side.
3. Addition - Addition operator perform addition of one to partial product.
The booth encoding can be simplified using the expressions follows

$$\text{Direction} = Y_{m+1};$$
$$\text{Addition} = Y_{m-1} \oplus Y_m;$$
$$\text{Shift} = Y_{m+1} \oplus Y_m;$$

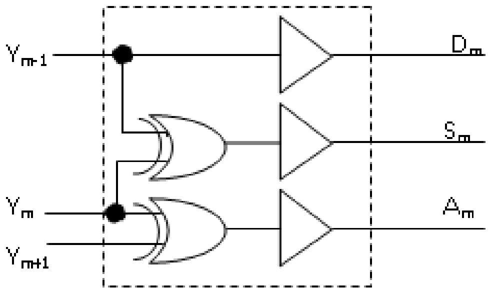


Fig 6. Booth Encoder Circuit

These signals are given to the partial product generator to produce the partial products based on the operator signal. Totally we are having 53 bits of mantissa, by grouping it as 3 bits; we will get the 27 groups of 3 bits inputs. So for 27 groups of bits all the 3 signals are generated and then it will trigger the partial product generator. 27 rows of partial products are generated according to the signals from the booth encoder.

5.2 Partial Product Generator

Partial products are the intermediate results in the multiplications which are added to produce the final product. In this design the partial products are generated based on the signals from booth encoder. Here the output from the booth encoder is acting as one of the inputs to PPG and another input is given by the second mantissa value. Based on the encoder input value, the partial product selector has to be considered. Based on these 3 bits of groups, the partial products are produced with help of partial product selectors [6] such as 0, +1,-1,+2,-2.

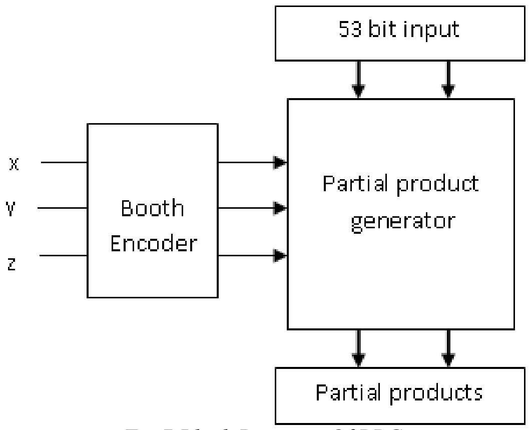


Fig 7.Block Diagram Of PPG

It illustrates how to calculate partial products from the bits of multiplicand B according to the values of the recoded digit.

Table 3.Relationship Of Partial-Product And Recoded Signed Bits

Multiplier bits	Selection
000	+0(0)
001	+1(p1)
010	+1(p1)
011	+2(p2)
100	-2(m2)
101	-1(m1)
110	-1(m1)
111	-0(0)

The computation of partial products given in Table 3 is simple:
For p1, the partial products equal the bits of B.
For p2, we obtain the partial products by a left shift of B.
For m1, we need to invert the bits of B and add the value 1 at the least significant bit.
For m2, we need to invert the bits of B, shift them left, and add the value 1 at the least significant bit.
For the encoded digit equal to 0, all partial products bits are equal to 0.

By doing these operations regarding to the table all the partial products in 27 rows is obtained, because of the 27 groups of bits of inputs to the partial product generator. These partial products are needed to be arranged in the proper format to get the correct result at the output side.

5.3 Wallace Tree Structure

Fast process for multiplication of two numbers was developed by Wallace [9].
Two step process is used to multiply two numbers:

(1) The bit products are formed.
(2) The bit product matrix is "reduced" to a two row matrix by using carry-save adders.
(3) The last two rows are summed using a fast carry-propagate adder to produce the product.

The Wallace-Tree binary adder is a usual building block in the implementation of the binary

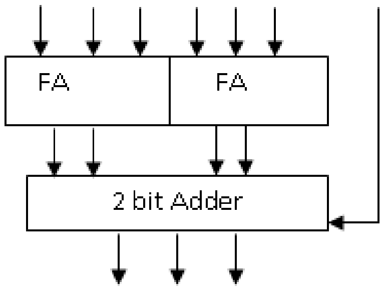


Fig 8. Wallace Element

multiplier, and is an integral element in the efficient implementation of high-speed binary multipliers.

5.4 XOR Based Conditional Select Adder

Instead of getting the final product directly, a special adder [10] can be used at the 2 rows of compressed output to get the final product. A conditional select adder having a carry generating unit which generates a carry of two n-bit input data units X_0-X_{n-1}, and Y_0-Y_{n-1}, and a sum generating unit which generates the sum of the input data provided.

The carry generating unit comprises

- a first input unit which receives predetermined data based on the input data X_i and Y_i;
- a second input unit which receives the initial carry;
- and a selection unit which receives the result of performing an XOR operation on the input data X_i and Y_i,

According to the XOR result, either predetermined data based on the input data X_i or Y_i input to the first input unit, or the initial carry input to this input unit is selected and output as a carry. The sum generating unit calculates a sum using the carry generated by the carry generating unit.

The main advantages are reducing power consumption, chip area reduction, reducing logic count, and delay time. This improved adder based on XOR is mainly proposed to minimize gate counts and critical path. There are fourteen XCSA (XOR based conditional select adder) blocks and a separated carry generation block are combined to make the 108 bit proposed adder structure. Each modularized XCSA consists of

- An 8-bit sum generator and
- A carry generator

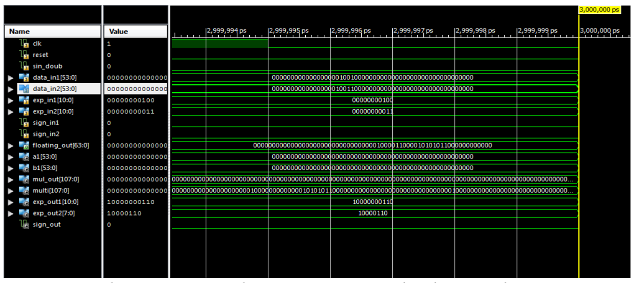
Instead of going with the architecture, the expressions can be used to make the operation as a simple one. The following expressions are describes how to determine a sum and a carry by XOR (A, B) [10].

Sum = $A_m \oplus B_m$;
Carry = if(($A_m \oplus B_m$) == 1) then cout = cin;
else if (($A_m \oplus B_m$) == 0) then cout = A_m;

By using these equations final sum and carry can be calculated. This final sum is the final product of mantissa multiplication. Due to this adder the speed is improved.

6. IMPLEMENTATION AND DISCUSSION

For implementation Xilinx ISE Design Suite 13.1with VHDL programming was used. Simulation process was done using ISIM tool .The results of a floating point multiplier with and without conditional select adder is shown. In order to evaluate the proposed architecture, a benchmark circuit namely a 4 tap FIR filter was implemented. Both single-precision and double-precision versions of each circuit were built in order to evaluate the multimode Floating point multiplier in both precision modes.



```
Name                 Value
clk                  1
reset                0
sin_doub             0
data_in1[53:0]       00000000000000
data_in2[53:0]       00000000000000
exp_in1[10:0]        00000000100
exp_in2[10:0]        00000000011
sign_in1             0
sign_in2             0
floating_out[63:0]   00000000000000
a1[53:0]             00000000000000
b1[53:0]             00000000000000
mul_out[107:0]       00000000000000
multi[107:0]         00000000000000
exp_out1[10:0]       10000000110
exp_out2[7:0]        10000110
sign_out             0
```

Fig 9.Single Precision Floating Point Multiplier Without XCSA

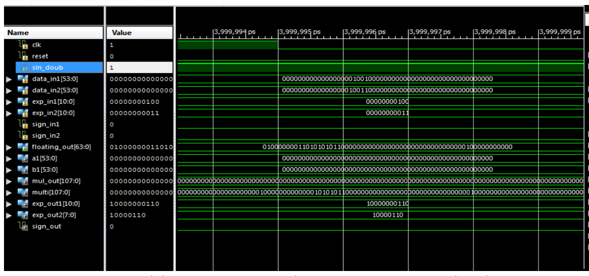


Fig 10.Double Precision Floating Point Multiplier Without XCSA

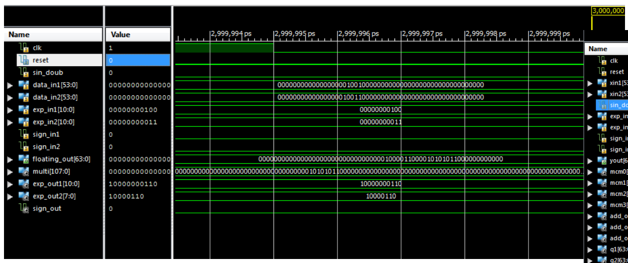


Fig 11.Single Precision Floating Point Multiplier With XCSA

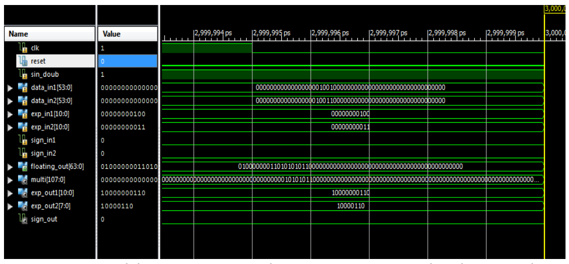


Fig 12.Double Precision Floating Point Multiplier With XCSA

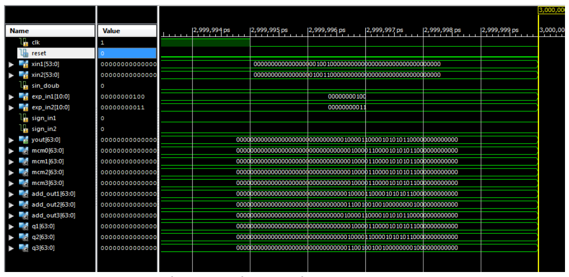


Fig 13.FIR Filter With Single Precision Non XCSA Floating Point Multiplier

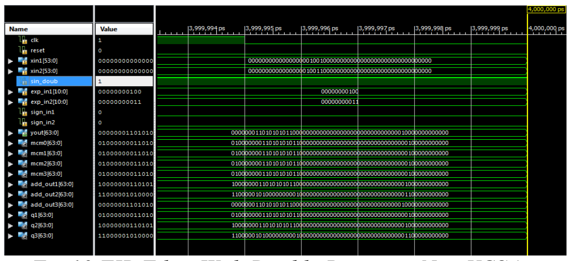


Fig 13.FIR Filter With Double Precision Non XCSA Floating Point Multiplier

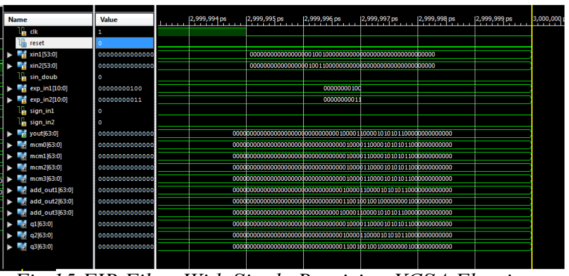


Fig 15.FIR Filter With Single Precision XCSA Floating Point Multiplier

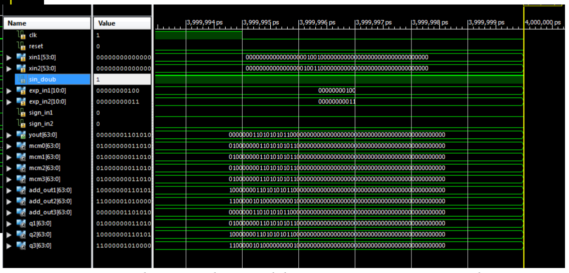


Fig 16.FIR Filter With Double Precision XCSA Floating Point Multiplier

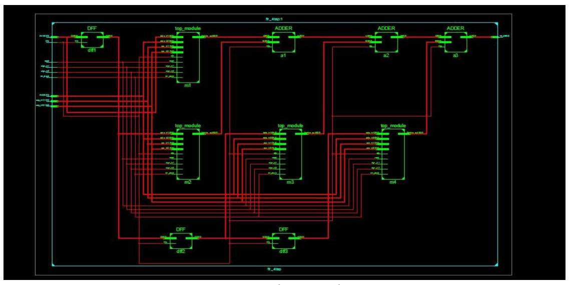


Fig 17.FIR Filter Schematic

Comparison of floating point multiplier with and without XCSA adder is shown in the following Table 4.

Table 4. Comparison Of Floating Point Multiplier

	Number of Slice FFs	Delay (ns)
Without XCSA	918 (9312) 9%	3.909
With XCSA	918(9312) 9%	3.878

Comparison of 4 tap FIR filter using floating point multiplier with and without XCSA adder is shown in the following Table 5.

Table 5. Comparison Of FIR Filter

	Number of Slice FFs	Delay (ns)
Without XCSA	3836 (9312) 41%	3.878
With XCSA	3832 (9312) 41%	3.726

Thus the results show that the floating point multiplier with XCSA adder provides around 4% better results with same slice FF counts.

7. CONCLUSION

This paper presented a flexible multimode floating-point multiplier for FPGAs. Each floating point multiplier can each perform double-precision operation or single-precision operation. Results show that the FPGA with embedded multimode FPUs provide considerable performance and area benefits in single-precision, double- precision, fixed-point, and integer applications.

REFRENCES:

[1] A. Akkas¸ and M. J. Schulte, “A quadruple precision and dual double precision floating-point multiplier,” in *Proc. Euromicro Symp. Digit. Syst. Des. (DSD)*, 2003, p. 76.

[2] G. Even, S. M. Mueller, and P.-M. Seidel, “A dual precision IEEE floating-point multiplier,” *Integr. VLSI J.*, vol. 29, no. 2, 2000, pp. 167–180.

[3] P. C. Diniz and G. Govindu, “Design of a field-programmable dual-precision floating-point arithmetic unit,” in *Proc. Int. Conf. Field Program. Logic Appl. (FPL)*, 2006, pp. 1–4.

[4] *IEEE Standard for Binary Floating-Point Arithmetic*, ANSI/IEEE Std 754, 1985.

[5] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design*, 3rd ed. San Francisco, CA: Morgan Kaufmann, ch. H.5, 2005.

[6] M. Nicolaidis and R. O. Duarte, “Fault-secure parity prediction booth multipliers,” *IEEE Des. Test*, vol. 16, no. 3, , Jul. 1999, pp. 90–101.

[7] W.-C. Yeh and C.-W. Jen, “High-speed booth encoded parallel multiplier design,” *IEEE Trans. Comput.*, vol. 49, no. 7, Jul.2000, pp. 692–701.

[8] A. D. Booth, “A signed binary multiplication technique,” *Quarterly J. Mechan. Appl. Math.*, vol. 4, 1951, pp. 236–240.

[9] C. S. Wallace, “A suggestion for a fast multiplier,” *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, Feb. 1964, , pp. 14–17.

[10] Ki-seon Cho, Jong-on Park, Jin-seok Hong and Goang-seog Choi “54x54-Bit radix-4 multiplier based on modified booth algorithm”, *ACM, GLSVLSI’03*, Washington, DC, USA, 2003, pp. 233-236.

[11] Padma Devi, AshimaGirdher and Balwinder Singh “Improved carry select adder with reduced area and low power consumption”, *International Journal of Computer Applications*, vol. 3, no.4, 2010.

[12] Sudharsana rani B. and Vijayakumarraju V. “Reducing the size of partial product array in two’s complement multipliers”, *International Journal of Logic and Computation (IJLP),* vol.5, issue 2, 2012, pp. 714-727.

[13] Vojin G. Oklobdzija “High-Speed VLSI arithmetic units: Adders and Multipliers”, sep. 1999

[14] Wen M.C., Wang S.J. and Lin Y.N. “Low-power parallel multiplier with column bypassing”, *Electronics letter*, vol. 41, no. 10, 2005.

[15] Yee Jern Chong and Sri Parameswaran “Configurable multimode floating point units for FPGAs*”, IEEE Trans. VLSI*, vol. 19, no. 11, Nov 2011, pp. 2033-2044.

[16]Y.Dou, S.Vassiliadis, G.Kuzmanov and G.Gaydadjiev,“64-bit floating point FPGA matrix multiplication,” in Proc.ACM/SIGDA13thInt.Symp.Field-Program.Gate Arrays,2005, pp.86–95.

[17]K.S.Hemmert and K.D.Underwood, “An analysis of the double-precision floating point FFT on FPGAs,” presented at the ACM Int. Symp. Field Program .Gate Arrays , Monterey,CA,Feb.2004.

[18]G.Govindu, S.Choi, V.K.Prasanna, V.Daga, S.Gangadharpalli, and V.Sridhar, “A high performance and energy efficient architecture for floating poin tbased LU decomposition on FPGAs, ”in Proc. 11th Reconfigurable Arch.Workshop(RAW),SantaFe,NM,Apr.2004, p.149a.

[19]M.de Lorimer and A.DeHon, “Floating point sparse matrix-vector multiply for FPGAs, ”in Proc. ACM Int. Symp. Field Program.GateArrays,Monterey,CA,Feb.2005,pp.75–85.

[20]ConveyComputerCorporation,“Conveycomputer,”Richardson,TX,2008–2010 [Online] . Available : http :// www. Convey computer.com/

[21]K.Underwood, “FPGAs vs. CPUs: Trends in peak floating-point performance, ”inProc.ACM/SIGDA12thInt.Symp.Field Program.GateArrays,Monterey,CA,Feb.2004,pp.171–180.

[22]M.J.Beauchamp, S.Hauck, and K.S.Hemmert,“ Embedded floating-point units in FPGAs, ”in Proc. IEEE Symp.Field Program.GateArrays(FPGA),2006,pp.12–20.

[23]C.H.Ho, P.H.W.Leong ,W.Luk,S.J.E.Wilton, and S.Lopez-Buedo,“ Virtual embedded blocks :A methodology for evaluating embedded elements in FPGAs, ”in Proc.IEEE Symp.Field-Program. Custom Comput. Mach.(FCCM),2006,pp.35–44.

[24]C.H.Ho,C.W.Yu,P.H.W.Leong,W.Luk,andS.J.E.Wilton,“Domain-specific hybrid FPGA :Architecture and floating point applications,” in Proc. Int .Conf .Field Program. Logic Appl.(FPL),2007,pp.196–201.