Design & Implementation of Single Precision Floating Point Multiplier using DADDA Architecture with FPGA

Aravind S.[1] Bharath RB.[2] Chinnaraj Nayak[3] Ganesh Shetti[4] Nithyashree S.[5]
[5]Assistant Professor
[1,2,3,4,5]Dr. AIT, India

Abstract— This paper work is devoted for the design and simulation of Single-Precision Floating Point Multiplier for signed and unsigned numbers using DADDA architecture. The multiplication operation is present in most parts of a digital system or digital computer, most notably in signal processing, graphics and scientific computation. With advances in technology, several techniques have been proposed to design multipliers, which offer high speed, low power consumption and requires lesser area. Thus making them suitable for various high speeds, low power compact VLSI implementations. The Design employs Combination of different adders and fast Compressors. The Carry Save Compressors (CSC) tree and the final Carry Propagate adder (CLA) chain is used to speed up the multiplier operation. Since signed and unsigned multiplication operation is performed by the same multiplier unit the required hardware and the chip area reduces and this in turn reduces power dissipation and cost of a system. Verilog coding of multiplier for signed and unsigned numbers using DADDA Algorithm for 32X32 bit multiplication following the Standard IEEE Format and their FPGA implementation by Xilinx Synthesis Tool on Spartan 6 kit have been done. The output has been displayed on LED of Spartan 6 kit. The multiplier structure is also designed in Cadence Software with 180nm technology.
Key words: Single-Precision Floating Point Multiplication, DADDA Reduction, Wallace, Modified Booth Encoding

I. INTRODUCTION

Multipliers are amidst the fundamental components in modern electronic systems that run complex calculations in both digital signal processors and general purpose processors. As the technology is improving, many researchers are trying to develop efficient multiplier designs that can offer high speed or Consume low power or requires less area or the combination of all these three in single multiplier unit. But for the portable devices power and area are the main constrains and should be minimized as much as possible. Such type of multiplier is implemented here. Initially a simple AND operation on each inputs bits generates the required partial products which are supposed to be added in the further steps. Then comes addition of these partial products. In the recent days Dadda or Wallace are been used for addition of partial products. But when we reconsider Wallace and Dadda multipliers it is proved that the hardware requirement for Dadda is less than the Wallace multiplier [11, 12]. Dadda reduction method performs faster addition of partial products [11]. The floating-point multiplier (FPM) is the major logical block in the Floating-Point unit (FPU) [2] [3] and ALU [13]. The IEEE-754 standard sets down specific rules and formats for any system that uses floating-point arithmetic [4]. The main reason to consider Single precision floating-point multiplier is that many standard FPUs support this format [5, 6]. This paper is divided into five sections. Basic floating-point multiplication Architecture is explained in section II. Section III provides the actual design of FPM. In section IV results, comparison and related discussions are provided. Conclusion and future work is given in section V.

II. FLOATING-POINT ARCHITECTURE

A. Standard IEEE Floating Point Representation

Our multiplier unit performs multiplication on Single precision floating-point data. i.e., the inputs and outputs to the unit are in standard IEEE-754 Single precision format [4]. The standard IEEE-754 single precision floating-point format consists of a 32-bit vector split into three sections as shown in Fig. 1. The Single precision floating-point number is calculated as shown in equation (1). Any floating-point number is interpreted as follows:

$$A=(-1)^{sign(A)} \times 1.fraction(A) \times 2^{exp(A)-bias} \quad (1)$$

To represent any floating-point number, the three fields are combined as follows

Sign (1-bit)	Exponent (8-bits)	Matissa (23-bits)

Fig. 1: IEEE-754 Single Precision (32-bit) Floating Point Format

B. Floating-Point Multiplication

The floating-point multiplication involves following steps: Assume that the two operands A and B are in IEEE-754 Single precision format, performing floating-point multiplication.

$$A \times B = (-1)^{exp(A)} (A_{man} \times 2^{exp(A)}) \times (1)^{exp(B)} (B_{man} \times 2^{exp(B)}) \quad (2)$$

Where "man" represents mantissa, "exp" represents exponent.

1) Computing the sign of the result as (Aexp ^ Bexp).
2) Multiplying the mantissa bits from A and B along with a bias bit "1" i.e, 23+1 bits.
3) Normalizing the product if needed.
4) Computing the exponent of the result as:
   exponent of the result = biased exponent (Aexp) + biased exponent (Bexp) – bias
5) Round the result to the number of input mantissa bits.

The block diagram of Single precision floating-point multiplier architecture is as shown in Fig. 2. The architecture contains a multiplier tree which multiplies two 24-bit numbers,(1 hidden bit i.e, the bias bit and 23 mantissa bits). Output from the multiplier tree is in carry save format and it is passed to a combined add/round stage, where the carry save product is added and rounded. Both the sign and exponent calculation logic runs simultaneously with the mantissa multiplication. To get the final exponent value, the exponent needs to be adjusted based on the rounding. Sometimes there is a chance of getting a wrong value for the exponent as the actual exponent value exceeds the bit range. In order to avoid this problem, in our design exception handling is included.

Whenever the bit range is high, an exception is generated while performing exponent logic or exponent adjust.

C. Floating Point Rounding

1) When rounding a "halfway" result to the nearest floating-point number, it picks the one that is even.
2) It includes the special values NaN, ∞, and −∞.
3) It uses denormal numbers to represent the result of computations whose value is less than 1.0 × 2^Emin (Emin is the minimum of exponent).
4) It rounds to nearest by default, but it also has three other rounding modes.
5) It has sophisticated facilities for handling exceptions.

To elaborate on (1), when operating on two floating-point numbers, the result is usually a number that cannot be exactly represented as another floating-point number. For example, in a floating-point system using base 10 and two significant digits, 6.1 × 0.5 = 3.05. This needs to be rounded to two digits. Should it be rounded to 3.0 or 3.1? In the IEEE standard, such halfway cases are rounded to the number whose low-order digit is even. That is, 3.05 rounds to 3.0, not 3.1. The standard actually has four rounding modes as discussed in the next section. The default is round to nearest, which rounds to an even number as it was just explained.

D. IEEE Rounding Modes

As per IEEE-754 standards there are four rounding modes as follows:

1) Round to nearest (RN)
2) Round to zero (RZ)
3) Round to positive infinity (RPI)
4) Round to negative infinity (RNI)

Implementation wise these rounding are further reduced to three. Round up (RU) with fix up, round to infinity, round to zero [7]. Round to nearest can be implemented as Round up with a fix up [7]. We are using RU in our floating-point arithmetic. Mathematically RU is given as follows

$$x = \begin{cases} \lceil x \rceil \; if \;\; x - \lfloor x \rfloor \geq 0.5 \\ \lfloor x \rfloor \qquad otherwise \end{cases}$$

The figure 2 below illustrates the implementation of Floating-point Multiplier Architecture.

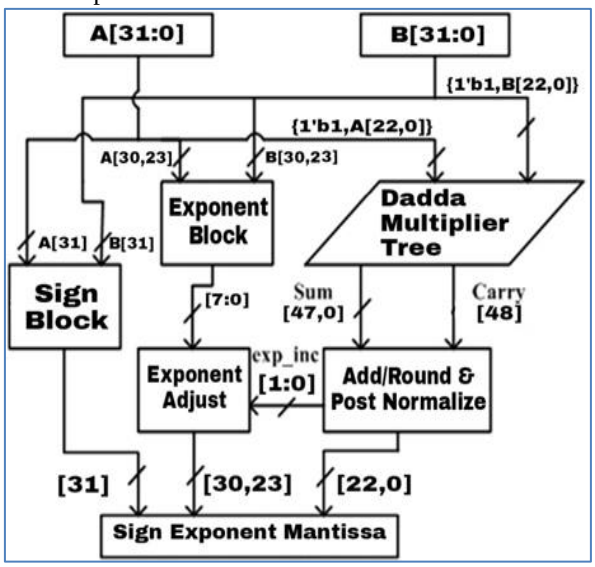

Fig. 2: Floating-Point Multiplier Architecture

III. LOATING-POINT MULTIPLIER USING DADDA ALGORITHM

A. Simple Multiplication Operation Involves Three Major Steps

1) Generation of partial products.
2) Reduce the generated partial products into two rows i.e., one row of final sums and one row of carries.
3) These final sums and carries are to be added to generate final result.

In this design Dadda algorithm is employed to reduce the generated partial products into one row of final sums and one row of carries. Finally these two rows are added using Carry propagate adders (CPA) chain.

B. Efficient Fast Compressor Designs

In the multiplication process, compressors are normally used to serve three major purposes, (i) to speed up the operation (ii) to reduce the number of stages in the partial product generation and (iii) to reduce the hardware requirement. Prior to the actual reduction of partial product it is important to design efficient fast Compressors and adders that adds the partial products generated.

The general block diagrams of different multiple input compressors are as shown in the below Figures. Basically such compressor designs came into existence to replace the usage of more number of full adders and reduce the complexity of the design.

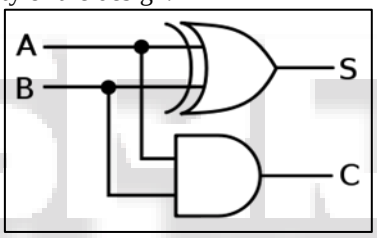

Fig. 3: half adder

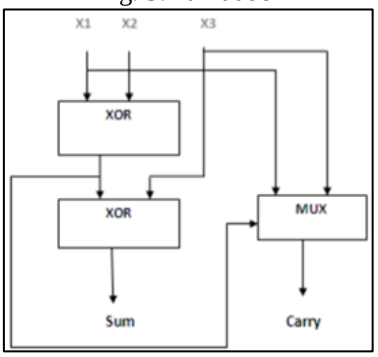

Fig. 4: (3:2) Compressor

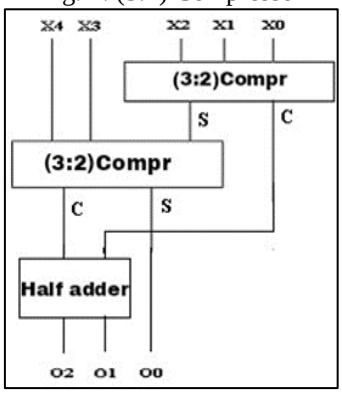

Fig. 5: (5:3) Compressor

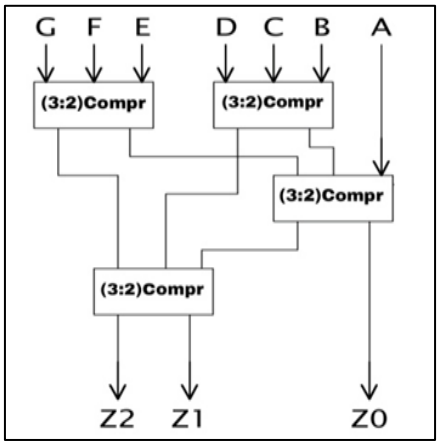

Fig. 6: (7:3) Compressor

C. Dadda Reduction Approach

After the generation of partial products, the partial products need to be added. Normally addition of these partial products consumes time. For this reason Dadda scheme is used to minimize the number of adder stages, by which delay can be optimized. Reduction of these partial products into two rows is done in stages using half adders and fast Compressors. The reduction in size of each stage is calculated by working back from the final stage. Each preceding stage height must be not greater than floor (1.5*successor height) [9]. Heights for the various stages can be 2, 3, 4, 6, 9, 13, 19, 28, 42, 63,94 etc., The Dadda reduction diagram for 8 bit by 8 bit multiplication is shown in Fig 7, this is performed in 3 stages.

Fig. 7: 8 bit by 8 bit DADDA Reduction

IV. RESULTS, COMPARISON & DISCUSSION

The simulation results of the floating-point Dadda multiplier shown in Fig. 8. Provides the synthesis results for the multiplier designed. The DADDA Floating-point multiplier has been compared with Wallace tree and simple Modified Booth Encoding Algorithm. In contrast to the Wallace reduction, Dadda algorithm does the least reduction necessary at each stage. Dadda reduction scheme uses lesser half adders and Compressors when compared to the Wallace reduction scheme. Thus Dadda multiplier's area and power is less than Wallace.

It is observed that with CMOS Circuit design technology, Modified Booth Algorithm requires the integration of 14,742 transistors using Booth encoder and decoder to generate 13 partial product rows. But Dadda architecture requires just 3,456 transistors using simple AND operation to generate 24 partial product rows. In conventional MBE, partial products are reduced using CSA trees but it consumes more time. To resolve this Dadda reduction scheme is used.

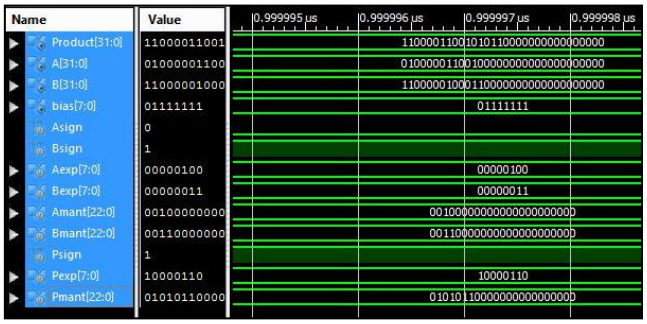

Fig. 8: Simulation Results of Dadda Multiplier

V. CONCLUSION & FUTURE SCOPE

This paper presents the design of a low power, high speed and area efficient single precision floating point multiplier using an Efficient DADDA Reduction Architecture. The design architecture has been compared with Modified Booth Encoding (MBE) and Wallace architectures. It is found that the multiplier has reduced power and area and it consumes comparatively very less power and Area. The comparison results in the previous sections prove that the design is efficient. This multiplier design is suitable for high performance floating point units or floating point multiply-add units of the co-processors.

As the future work, this work can be extended to achieve better speeds by using Residue Number System [1]. And also to develop and design an efficient double precision floating-point multipliers Architecture.

REFERENCES

[1] Dhanabal R, Sarat Kumar Sahoo, Barathi V, N.R.Samhitha, Neethu Acha Cherian, Pretty Mariam Jacob, "Implementation of Floating Point MAC using Residue Number System", Journal of Theoretical and Applied Information Technology, vol. 62, no. 2, April 2014.

[2] Dhanabal R, Bharathi V, Shilpa K, Sujana D.V and Sahoo S.K, "Design and Implementation of Low Power Floating Point Arithmetic Unit", International Journal of Applied Engineering Research, ISSN 0973-4562, vol. 9, no. 3, pp. 339-346, 2014.

[3] Ushasree G, Dhanabal R, Sarat kumar sahoo, "VLSI Implementation of a High Speed Single Precision Floating Point Unit Using Verilog", Proceedings of IEEE Conference on Information and Communication Technologies (ICT 2013), pp. 803-808, 2013

[4] IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985, Reaffirmed Dec. 6, 1990, Inc., 1985.
[5] R. K. Montoye, E. Hokenek, and S. L. Runyon, "Design of the IBM RISC System/6000 floating-point execution unit," IBM J. Res. Development, vol. 34, pp. 59–70, 1990.
[6] E. Hokenek, R. Montoye, and P. W. Cook, "Second-generation RISC floating point with multiply-add fused," IEEE J. Solid-State Circuits, vol. 25, no. 5, pp. 1207–1213, Oct. 1990.
[7] N. Quach, N. Takagi, and M. Flynn, "On fast IEEE rounding," Stanford Univ., Stanford, CA, Tech. Rep. CSL-TR-91-459, Jan. 1991.
[8] L. Dadda, "Some schemes for parallel multipliers," IEEE Transactions on Computers, vol. 13, pp.14-17, 1964.
[9] Waters. R. S, Swartzlander. E. E, "A Reduced Complexity Wallace Multiplier Reduction," Computers, IEEE Transactions on, vol.59, no.8, pp.1134-1137, Aug. 2010.
[10] Giorgos Dimitrakopoulos and Dimitris Nikolos, "High-Speed Parallel-Prefix VLSI Ling Adders," IEEE Transactions on Computers, vol. 54, no. 2, pp. 225-231, Feb 2005.
[11] K.C. Bickerstaff, E. E. Swartzlander, and M. J. Schulte, "Analysis of column compression multipliers," in proceedings of the 15th IEEE symposium on Computer Arithmetic, pp. 33-39, June 2001.
[12] W. J. Townsend, E. E. Swartzlander, and J. A. Abraham, "A comparison of Dadda and Wallace multiplier delays," in Advanced Signal Processing Algorithms, Architectures, and Implementations XIII, vol. 5205 of Proceedings of the SPIE, pp. 552–560, August 2003.
[13] Dhanabal R, Bharathi V, Salim S, Thomas B, Soman H, and Sahoo.S.K, "Design of 16-bit low power ALU-DBGPU", International Journal of Engineering and Technology, vol. 5 no. 3, pp. 2172 – 2180, Jun 2013.
[14] D. Booth, "A Signed Binary Multiplication Technique," Quarterly J. Mechanical and Applied Math., vol. 4, pp.236-240, 1951.

