2025 3rd International Conference on Device Intelligence, Computing and Communication Technologies (DICCT) | 979-8-3315-4335-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/DICCT64131.2025.10986442

FPGA Implementation of Floating-Point Multiplier Employing Carry-Lookahead Adders

Abhay Sharma
Graphic Era (Deemed to be University)
Dehradun, India
abhayep03@ieee.org

Md. Irfanul Hasan
Graphic Era (Deemed to be University)
Dehradun, India
irfanhasan25@rediffmail.com

Devesh Tiwari
Graphic Era Hill University
Dehradun, India
deveshtiwari@gehu.ac.in

***Abstract*—For hardware implementation of digital signal processing algorithms, the key components are multipliers. The design of multipliers plays an important role in overall system architecture. For algorithms requiring a dynamic range of data, floating-point representation is favoured over fixed-point. Nevertheless, floating-point multipliers provide difficulties for designers because of the large latency and space that they need. In this work, we suggest a more efficient method of floating-point multiplication with respect to both time and space. The bottleneck in the floating-point multiplier design is the Mantissa multiplication which is implemented through fast tree multipliers. Instead of using half-adders and full-adders as in conventional Wallace or Dadda tree multipliers, Carry-lookahead adders are used as compressors in the proposed structure, which resulted in a better partial product reduction in mantissa multiplication. The proposed design is validated through FPGA implementation using Verilog HDL description. It was observed that an improvement of 8.5% with respect to Wallace and an improvement of 9.9% with respect to Dadda is obtained in terms of latency.**

***Index Terms*—Floating-point, FPGA, Tree multipliers, Carry-lookahead**

I. Introduction

Multiplier Design is of concern in high performance computing structures such as filter design [1] as it is the second most used operation after addition. The speed with which a multiplication can be completed is sometimes the deciding factor in the success or failure of a processing job [2]. Applications' needs for dynamic range and/or accuracy dictate which type of data-type to be employed for multiplier design. The range of values that may be represented without changing the bit-width is expanded in floating-point format. In addition, it avoids the overflow/underflow issues common to fixed-point digital designs [3].

Floating point formats may be generally classified according to the IEEE-754 standard as either 32-bit single precision or 64-bit double precision. In the vast majority of applications, a format with 32 bits is enough to accommodate the necessary range of numbers to be represented. As a result, the 32-bit floating point multiplier technique and the problems associated with its implementation were explored, and it was discovered that the mantissa multiplication unit is the component with the worst performance [4]–[13].

In general, multiplication of two binary numbers takes place in three phases. Firstly, partial product generation through logical and operation. Secondly, partial products are reduced through addition. In the last, final stage of addition generates the required product. The reduction phase takes the maximum computation time. Various methods have been proposed such as Wallace Tree multipliers [14] or Dadda Tree multipliers [15] for increasing efficiency in the reduction phase. In both the methods, full adders and half adders are used in the reduction phase. The maximum reduction ratio obtained is 3:2. It implies that 3 bits are reduced to 2 bits after passing partial products to single stage of full adders in reduction phase. In [16], it is shown that by using the expanded booth wallace method, one may anticipate and round off the partial products generated by booth coding using the symbolic expansion concept. Additionally, one can optimize the partial products generated by single-precision floating-point multiplication and the accumulation of partial products. Units for efficient mantissa multiplication using combined Wallace and Dadda structure is proposed in [17].

This work shows that a higher reduction ratio of 9:5 can be obtained through carry-lookahead adders. The proposed scheme reduces the delay in the reduction phase. Since in floating-point multiplication significant cost is associated with 24-bits by 24-bits mantissa multiplication, employing carry-lookahead adders will improve the overall performance.

II. Floating-Point Multiplication

According to the most up-to-date IEEE standard for floating-point arithmetic, different floating-point formats may represent different subsets of floating-point data depending on their radix, precision, and exponent range. Floating-point numbers are written as $(-1)^{sign} \times b^{exponent} \times mantissa$, where radix is 2 for binary and 10 for decimal, and precision refers to the amount of digits in the mantissa [18].

In the 32-bit binary format, the sign bit (S) indicates whether the number is positive or negative, the next 8 bits represent the biased exponent ($E = e + bias$), where e is the actual exponent and the bias value is 127, and the remaining 23 bits represent the mantissa. The floating-point format is depicted in Fig. 1.

The floating-point multiplication can be elaborated in the following steps:

1) Sign bits of both multiplier and multiplicand are passed through the exclusive-or operation for getting the sign bit of product.

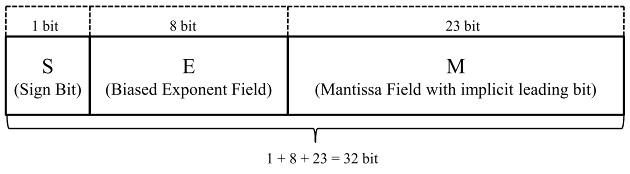


Fig. 1. 32 bit floating-point format

2) 8-bit exponents of both the multiplier and the multiplicand are added.
3) Mantissa of both the multiplier and multiplicand are multiplied.
4) Mantissa multiplication is followed by rounding and normalization to update the final product exponent.

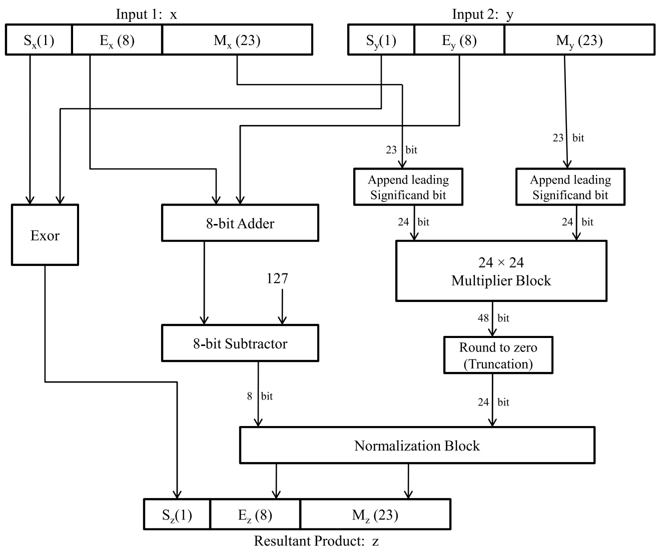


Fig. 2. 32 bit floating-point multiplication

Fig. 2 depicts a block schematic of the aforementioned procedures in action. The calculation of the sign bit, the addition of exponents, and the multiplication of mantissas are all operations that may be performed in parallel. However, the product from the mantissa multiplication must be evaluated before normalisation and rounding can be performed. Rounding to zero works nicely for most digital signal processing needs. So, our approach takes this into account by preserving the top 24 bits of the 48 bits product formed by the 24×24 mantissa multiplication and emitting the bottom 24 bits.The exponent is modified by finding the location of the leading 1 in the product and then either increasing or decreasing it. The radix point will be shifted to the left with an increase in the exponent, or to the right with a decrease in the exponent, depending on the detected leading position.

III. Mantissa Multiplication

A 24-bit unsigned multiplier is used for the mantissa multiplication. There are three sections to it. Firstly, logical AND operator are employed for generating partial products. In the second phase, partial product matrix is reduced to 2 rows, namely, sum row and carry row. Finally, these 2 rows are accumulated to obtain the product. Partial-product reduction often requires tree structures with parallel adders and consumes the most time. After the 24×24 matrix has been reduced to only two rows by the reduction tree, the proper ripple carry adder (RCA) may be utilised to compute the final result. The existing methods employ full adders and half adders for reducing the partial products. Such tree multipliers can be better visualized using dot notation as depicted in Fig. 3. In dot representation, bit positions instead of bit values are important, hence, dot are used to indicate bit positions.

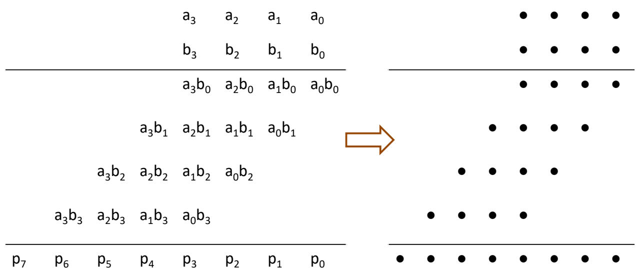

a_3 a_2 a_1 a_0
b_3 b_2 b_1 b_0
a_3b_0 a_2b_0 a_1b_0 a_0b_0
a_3b_1 a_2b_1 a_1b_1 a_0b_1
a_3b_2 a_2b_2 a_1b_2 a_0b_2
a_3b_3 a_2b_3 a_1b_3 a_0b_3
p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0

Fig. 3. Dot Diagram

The wallace tree multiplication algorithm is summarized as follows:

1) After the partial product array has been created, the rows that are next to each other are gathered together into groups of three that do not overlap.
2) It is possible to decrease each set of three rows by.
   a) Each column containing three bits full-adder is employed.
   b) Each column containing two bits half-adder is employed.
   c) Each column of single bit is passed to next stage.
3) Each subsequent step uses same reduction strategy until there are only two rows left.
4) A Carry Propogation Adder is employed to the remaining last two rows.

Fig. 4 illustrates the above algorithm for 4-bit by 4-bit multiplication.

The speed of multiplication can be increased by reducing the number of operands to be added i.e. number of partial products or by adding the operands (partial products) faster. The number of operands in case of mantissa multiplication is fixed to 24 rows of partial products. To speed up the reduction of partial products, this work propose using carry look-ahead adders. A 4-bit carry look-ahead adder (CLA4) as shown in Fig. 5 may take in a maximum of 9 PP bits and output a maximum of 5 PP bits, for a ratio of 1.8 to 1. Due to the fact that CLA4s are quicker at PP reduction than full adders, it is possible to build a multiplier with fewer reduction steps than Wallace/Dadda. When using a CLA with an input size more than 4 bits, the reduction step's

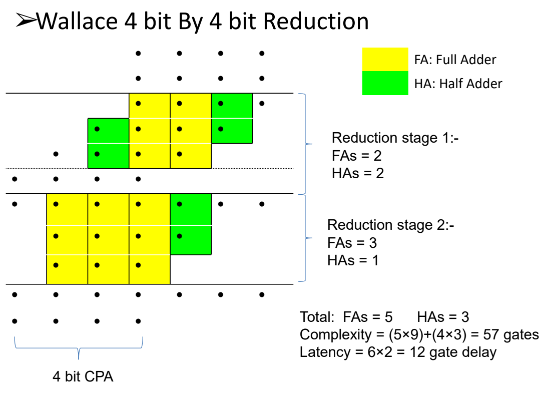


Fig. 4. Wallace 4 bit By 4 bit Reduction

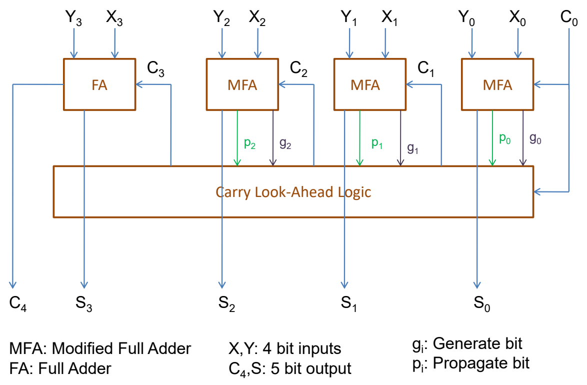


Fig. 5. 4 bit carry lookahead adder

performance begins to degrade. Because delays in CLAs with input sizes larger than four are more than or equal to ten gate delays, this is the case. This completely nullifies the benefit of increasing the input size. Fig. 6 describes the reduction of 8×8 partial product tree. In case of wallace/dadda, the number of reduction stages for 8×8 multiplier is 4, whereas for CLA4 based structure the reduction is accomplished in 3 stages. Due to less number of reduction stages, the speed of CLA4 based multiplier will be faster as compared to wallace/dadda multiplier. Using a high concentration of CLA4s, which has a greater reduction ratio, yields the greatest result. However, CLAs weren't without their issues. After the first round of simplification, the CLA multiplier took on a less regular shape. The placement of CLA4s using a greedy strategy is ineffective in several situations. An additional delay step may occur depending on the location of a particular CLA4. This means that after the first step, CLA4 is not consistently placed.

Due to parallelism in CLA4 structure, latency is 6 gate delays which is similar to full adder latency. However, the complexity of CLA4 is quit high in comparison to full adders. Fig. 7 provides the comparison in terms of gate delays for different multiplier size. From the above comparison it is evident that using carry look-ahead adders in reduction phase reduces the number of stages and hence the overall latency. Table I indicates the number of components utilized for designing multipliers of different sizes.

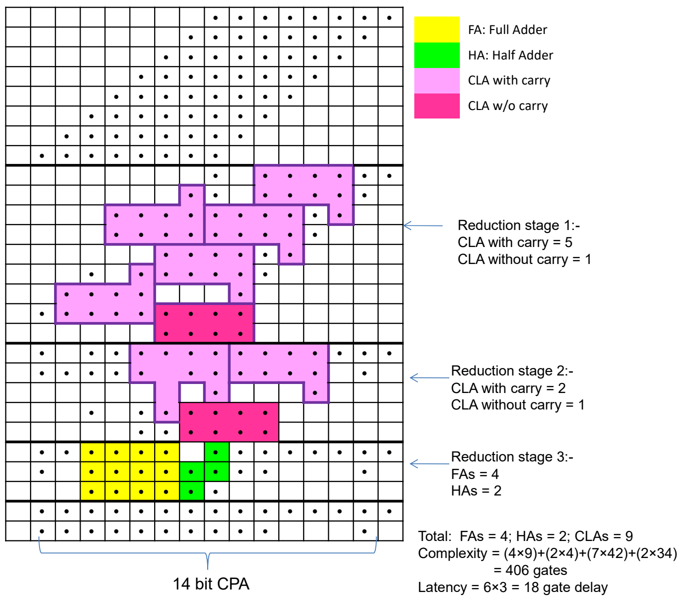


Fig. 6. 8 bit by 8 bit reduction using CLAs

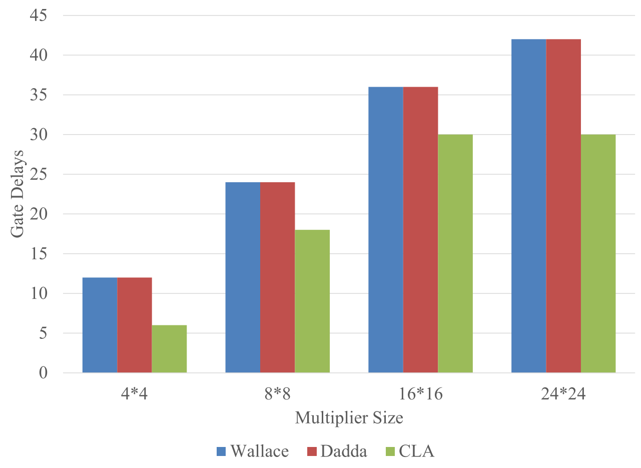


Fig. 7. Latency Comparison

TABLE I
UNIT UTILIZATION CLA BASED MULTIPLIER

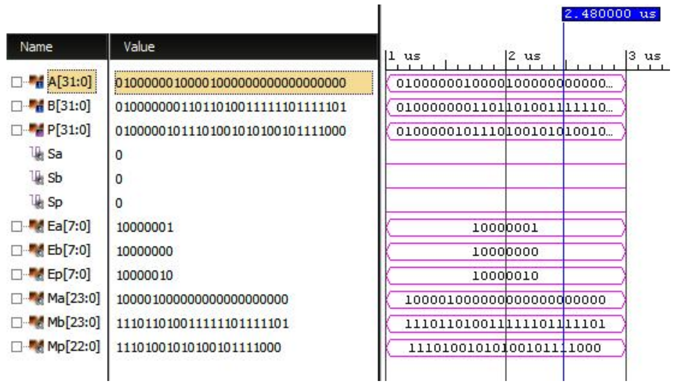
Units used	Size		
	*8*8*	*16*16*	*24*24*
Carry Look-ahead Adder	9	43	131
Full Adder	4	27	29
Half Adder	2	39	22

IV. FPGA IMPLEMENTATION

The field programmable gate array (FPGA) has become a prominent hardware platform for prototyping digital systems in contemporary times. Hardware description languages

(HDLs), such as Verilog and VHDL, are used in the description and modeling of digital systems with the specific aim of targeting a designated field-programmable gate array (FPGA) device. In this study, the Verilog Hardware Description Language (HDL) is used to develop a model for a floating-point multiplier design. Subsequently, synthesis is conducted to convert the HDL representation into generic gate-level components using Xilinx Vivado 14.7, with a specific focus on targeting the Artix 7 Field-Programmable Gate Array (FPGA).

The test-bench simulation waveform is presented in Fig. 8. Two numbers 4.125 and 3.707 were multiplied to obtain the result 15.291375.



Name	Value
A[31:0]	01000000100001000000000000000000
B[31:0]	01000000011011010011111101111101
P[31:0]	01000001011101001010100101111000
Sa	0
Sb	0
Sp	0
Ea[7:0]	10000001
Eb[7:0]	10000000
Ep[7:0]	10000010
Ma[23:0]	100001000000000000000000
Mb[23:0]	111011010011111101111101
Mp[22:0]	11101001010100101111000

Fig. 8. Simulation Output Waveforms

After behavioral simulation, synthesis and implementation were done to generate the implementation report presented in Table II. In order to compare the proposed structure with the existing conventional techniques, Verilog description for floating-point multiplier employing Wallace tree and Dadda tree based Mantissa multiplication were also implemented. Report shows an improvement in terms of path delay which in turn will reduce the overall latency.

TABLE II
FPGA IMPLEMENTATION REPORT

Mantissa Multiplier Using	Max. Path Delay	Device Utilization
Wallace Tree	36.53ns	852
Dadda Tree	37.13ns	857
CLA Tree (Proposed)	33.42ns	925

V. CONCLUSION

This work proposes a more efficient method of floating-point multiplication for digital signal processing applications in terms of hardware implementation. Instead of using half-adders and full-adders in conventional Wallace or Dadda tree multipliers, Carry-lookahead adders are used as compressors. This results in a better partial product reduction in mantissa multiplication, reducing the latency. It was observed that an improvement of 8.5% with respect to Wallace and an improvement of 9.9% with respect to Dadda is obtained in path delay through Verilog implementation. However, the device utilization was increased which is trade-off for latency. This approach is more suitable for handling dynamic data ranges.

REFERENCES

[1] M. Gupta, M. Kansal, S. Thyagarajan, P.S. Chauhan,D.K. Upadhyay,"Design and Analysis of Non-uniform Transmission Line Based Dual-Band Bandpass Filter," Advances in VLSI, Communication, and Signal Processing. Lecture Notes in Electrical Engineering, vol 911, Springer, Singapore, 2022, pp. 507–518.

[2] S. Abraham, S. Kaur and S. Singh, "Study of various high speed multipliers," 2015 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, India, 2015, pp. 1-5.

[3] C. Inacio and D. Ombres, "The DSP decision: fixed point or floating?,"in IEEE Spectrum, vol. 33, no. 9, pp. 72-74, Sept. 1996.

[4] D. Goldberg, "What every computer scientist should know about floating-point arithmetic." ACM computing surveys (CSUR) vol. 23, no. 1, pp. 5-48, 1991.

[5] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," 2013 18th IEEE European Test Symposium (ETS), Avignon, pp. 1-6,27-30 May 2013.

[6] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi and J. Han, "A comparative evaluation of approximate multipliers," 2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Beijing, pp. 191-196, 18-20 July 2016.

[7] IEEE Standard for Floating-Point Arithmetic," in IEEE Std 754-2008 , vol., no., pp.1-70, Aug. 29 2008.

[8] Behrooz Prahami, Computer Arithmetic Algorithms and Hardware Designs, Oxford.

[9] K. V. Gowreesrinivas and P. Samundiswary, "Comparative analysis of single precision floating point multiplication using compressor techniques," IEEE International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Chennai, pp. 2428-2433, 22-24 March 2017.

[10] M. Mehta, V. Parmar, E. Swartzlander, "High-speed multiplier design using multi-input counter and compressor circuits", Computer Arithmetic 1991. Proceedings. 10th IEEE Symposium on, pp. 43-50, Jun 1991.

[11] S. Asif and Y. Kong, "Design of an algorithmic Wallace multiplier using high speed counters," International Conference on Computer Engineering and Systems (ICCES), Cairo, pp. 133-138,23-24 Dec. 2015.

[12] Abhay Sharma,"FPGA Implementation of a High Speed Multiplier Employing Carry Lookahead Adders in Reduction Phase", International Journal of Computer Applications vol 116 number,17 pp. 27-31, April 2015.

[13] K. Arun and K. Srivatsan, "A binary high speed floating point multiplier," IEEE International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2), Chennai, pp. 316-321, 23-25 March 2017.

[14] C.S. Wallace, "A suggestion for fast multiplier," IEEE trans. Electron. Comput., vol. EC-13, pp. 14-17, 1964.

[15] L. Dadda, "Some schemes for parallel multipliers", Alta Frequenza, vol. 34, pp. 349-356, 1965.

[16] Na Bai, Hang Li, Jiming Lv, Shuai Yang, Yaohua Xu, "Logic Design and Power Optimization of Floating-Point Multipliers", Computational Intelligence and Neuroscience, vol. 2022, Article ID 6949846, 2022.

[17] Anuradha, Sujit Kumar Patel, Subodh Kumar Singhal, "An area-delay efficient single-precision floating-point multiplier for VLSI systems", Microprocessors and Microsystems, Vol. 98, 2023.

[18] IEEE Standard for Floating-Point Arithmetic," in IEEE Std 754-2008 , vol., no., pp.1-70, Aug. 29 2008.