Volume 18 · Issue 1
January-April 2021
Pages 1-9

CULCYT
CULTURA CIENTÍFICA Y TECNOLÓGICA
ISSN (electrónico) 2007-0411

DOI: 10.20983/culcyt.2021.1.2.4

RESEARCH ARTICLE

VLSI Design and Comparative Analysis of Several Types of Fixed and Simple Precision Floating Point Multipliers

Diseño VLSI y Análisis Comparativo de Varios Tipos de Multiplicadores de Punto Fijo y Punto Flotante de Precisión Simple

Abimael Jiménez-Pérez[1], Marco Antonio Gurrola-Navarro[2], Víctor Manuel Valenzuela-De la Cruz[3], José Antonio Muñoz-Gómez[2], Omar Aguilar-Loreto[2]

[1] Universidad Autónoma de Ciudad Juárez
[2] Universidad de Guadalajara
[3] Intel Guadalajara

ABSTRACT

Multiplication is an arithmetic operation that has a meaningful impact on the performance of several real-life applications, such as digital signal and image processing. Analysis and comparison of different types of fixed-point multipliers such as Wallace tree, array, and Booth-2 with truncated and non-truncated versions were included in this design. Fixed-point multipliers were used to design floating-point multipliers through a hardware description language. As a result, area and speed values were analyzed. Booth-2 fixed multiplier with truncation and RCA adders present both the longest delay and the largest area consumption. Wallace tree floating-point multiplier required the smallest area and the shortest delay. The 8-bit versions of fixed-point multipliers were physically synthesized, using the Alliance tools, to obtain the layout of the circuits. The integrated circuits were successfully fabricated in a 0.5-µm CMOS technology.

KEYWORDS: VLSI integrated circuit; VHDL; Booth-2; Wallace tree; floating-point.

RESUMEN

La multiplicación es una operación aritmética que tiene un impacto significativo en el rendimiento de varias aplicaciones de la vida real, como el procesamiento de imágenes y señales digitales. En este trabajo se analizan y comparan de diferentes tipos de multiplicadores de punto fijo, como árbol de Wallace, Arreglo y Booth-2 con versiones truncadas y sin truncar. Los multiplicadores de punto fijo se utilizaron para diseñar multiplicadores de punto flotante a través de un lenguaje de descripción de hardware. Como resultado, se analizaron los valores de área y retardo. El multiplicador de punto fijo Booth-2 con truncamiento y sumadores RCA presentó tanto el mayor retardo como el mayor consumo de área. El multiplicador de punto flotante del árbol de Wallace requería el área más pequeña y el retraso más corto. Las versiones de 8 bits de los multiplicadores de punto fijo se sintetizaron físicamente para obtener el *layout*. Los circuitos integrados se fabricaron con éxito en una tecnología CMOS de 0.5 µm.

PALABRAS CLAVE: Circuito integrado VLSI; VHDL; Booth-2; árbol de Wallace; punto flotante.

Corresponding author: Abimael Jiménez Pérez
Institution: Universidad Autónoma de Ciudad Juárez / Instituto de Ingeniería y Tecnología
Address: Av. Del Charro núm. 450, col. Partido Romero, Ciudad Juárez, Chihuahua, México, C. P. 32310
E-mail: abimael.jimenez@uacj.mx

Manuscript received: February 10, 2021; accepted: April 25, 2021. Date of publication: April 30, 2021.

open access UACJ
PEER REVIEW

I. INTRODUCTION

Multiplier is an essential component in every digital signal processing, image processing, and computer vision applications [1]. The need for greater functionality and real-time applications demands revolutionary changes in the design process of a Very Large-Scale Integration (VLSI) Integrated Circuit (IC). Therefore, the development of high-speed computational hardware, such as multipliers, is a major concern in the current scenario [2]–[3]. The most important design criteria of these kinds of components are speed, power, and area consumption. Various research efforts have been carried out in literature to obtain efficient multiplier and adder architectures [2]–[5].

The state of the art of VLSI designs had focused mainly on the reduction of the area, but in the last decade the focus changed mainly to speed and power consumption. A high-speed requirement causes increased circuit complexity, increasing both the number of transistors of the circuit and the power consumption [5]. However, it is possible to improve the performance of multipliers through design techniques and logic architectures [6]–[9].

FLOATING-POINT NOTATION

Floating-point numbers are formally defined by the IEEE 754:2008 standard [10], where three types of binary numbers are specified 32-, 64-, and 128-bit.

The format of a 32-bit binary number is shown in Figure 1 and it is demonstrated that a floating-point number can be represented in single or double precision under this standard. In this work, we will focus only on simple precision. Nevertheless, the concepts and schemes could be extended to greater precisions.

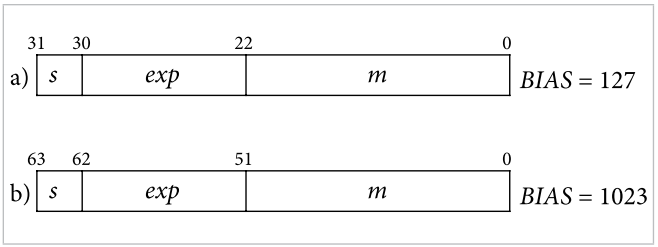

Figure 1. Number of 32 bits defined by IEEE-754:2008, a) single precision, b) double precision.

For a number n in floating-point, a bit is added to the fraction to form mantissa (or significand). The normalized number can be represented by

$$n = (-1)^{s} \cdot 1 \cdot m \cdot 2^{(exp-BIAS)} \quad (1)$$

where s is the sign bit, m is the mantissa and exp is a positive integer power with $BIAS = 127$.

In addition to the normal format, there are special formats for exp and m, such as zero, denormalized, infinite, and no number. Further details about the rules of binary numbers can be found in [11].

TRUNCATION METHOD

In a product without truncation all partial products are obtained and added. Therefore, there is no truncation error; except for the intrinsic error when the data is truncated in the floating-point representation.

When truncation is considered, an error is generated, losing accuracy in the result. However, it allows the reduction of components in the hardware architecture. A truncated product omits the calculation of the partial products of least significant bits (LSB). In the truncated versions of multipliers, n–1 bits are truncated. This is because the extra bit permits the result to be shifted one position to the left in the normalization of m, if necessary. In this work, round-to-zero truncation is used, in which n LSB are neglected [12].

In this paper, we demonstrate the successful design and comparative analysis of three different designs of fixed-point multipliers (Array, Wallace Tree, and Booth-2), which were used to design simple precision floating-point multipliers. The details of this work are presented in the following sections.

II. METHODOLOGY

FLOATING-POINT MULTIPLIER ARCHITECTURE

As shown in Figure 2, in the design of a floating-point multiplier, 24-bit fixed-point (FP) multiplier, and modules to normalize m, handle exponents and verification of exceptions (overflow and underflow) are required. The proposed architecture is able to determine exceptions in the result and represents the data in the IEEE-

754: 2008 standard. However, it requires the normalization of the numbers first.

NORMALIZATION, EXPONENTS, AND EXCEPTIONS

Initially, the 32-bit input numbers are separated into signals to distinguish the sign *s*, mantissa m and exponents *exp* as shown in Figure 2. Then, the new sign bit is calculated with an XOR. The mantissas are multiplied with the 24-bit FP multiplier. Then the exponents are verified to detect a zero or error condition. After that, the 25 most significant -s (MSB) of the FP multiplication result (with or without truncation) are taken and, *m* is normalized if necessary. Finally, with the normalized numbers and according to Table I, *m* is determined. Accordingly, the bits for the new *m* will depend on the value of the MSB.

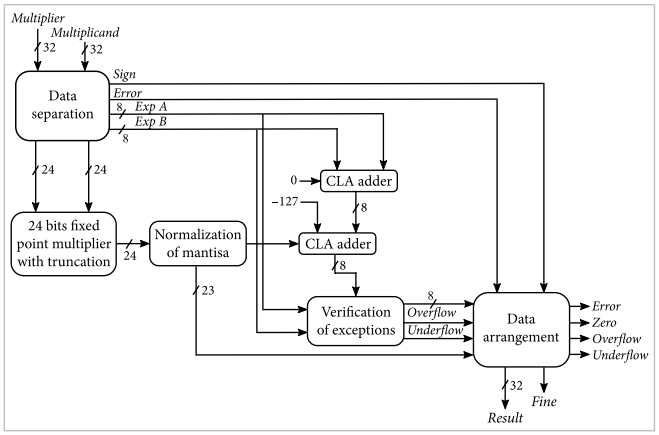

Figure 2. Architecture for floating-point multipliers.

TABLE I
NORMALIZATION OF MANTISSA

OBTAINED	REQUIRED	ACTION
10.000…	1.0000…	Displace *m* to the right
01.000…	No change	No change

For the exponents, corresponding to the multiplier and multiplicand are added. Then the *BIAS* value is subtracted to the result as shown in Figure 2. There is the possibility of adding a unit in the input carry of the adder, as a result of the normalization of m.

The resulting value of *exp* is verified by overflow or underflow conditions. Overflow occurs when the calculation generates a transition from a value ≤254 to a value ≤255, 0 or greater. Underflow occurs when the calculation generates a transition from a value 1≥ to a value 0, 255 or less. The flowchart to determine these exceptions is shown in Figure 3.

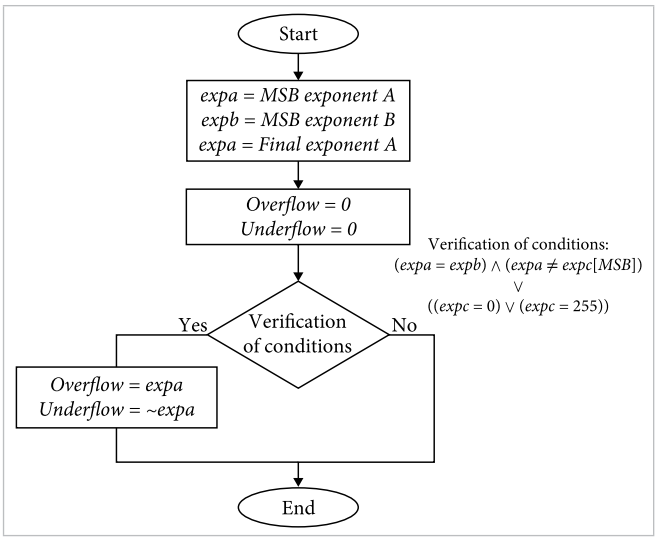

Figure 3. Flowchart for verification of exceptions.

TYPES OF MULTIPLIERS DESIGNED

To compare the performance of different floating-point multiplier architectures, three architectures were used to implement FP multipliers. The first is a typical array multiplier [13]. The second uses the Wallace tree technique [14] and the third is the Booth-2 [15]–[16]. Each FP multiplier was designed in 8-, 16-, 24-, and 32-bit, with Ripple Carry Adder (RCA) and Carry Look-Ahead (CLA) adders, as well as versions with and without truncation. However, it is important to mention that only 24-bit versions are used floating-point multipliers. Table II shows all the FP multipliers designed.

TABLE II
TYPES OF FP MULTIPLIERS DESIGNED

MULTIPLIER	NUMBER OF BITS	TRUNCATED	ADDERS TYPE
Array	8, 16, 24, and 32	No	RCA/CLA
		Yes	RCA/CLA
Wallace tree	8, 16, 24, and 32	No	RCA/CLA
		Yes	RCA/CLA
Booth-2	8, 16, 24, and 32	No	RCA/CLA
		Yes	RCA/CLA

ARRAY MULTIPLIER

Array multiplier performs multiplication of two numbers based on the shift and adds method as shown in

Cultura Científica y Tecnológica • Vol. 18, Issue 1
Pages 1-9 • January-April 2021 CULCYT

Figure 4. Even though it has a very regular and systematic structure, its delay becomes very large for a large word length [13]. First, all partial products with AND gates are obtained. The set of partial products $(x_i \cdot y)$ with even position (0, 2, 4, ..., n) will have a zero weight in its greatest weight position. Meanwhile, partial products $(x_i \cdot y)$ with odd position (1, 3, 5, ..., m) will have a zero in the least weight position.

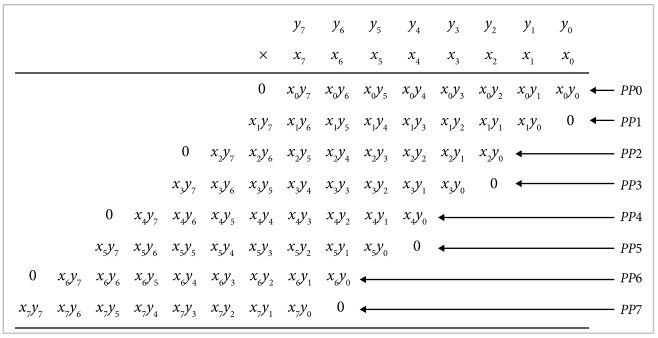

Figure 4. Array FP multiplier of 8 bits (non-truncated).

Then, the sums of the partial products are carried out as shown in Figure 5. The sets of partial products are added together with a shift to the right in the even sets and to the left in the odd ones. The number of zeroes added is equal to 2^{phase}. This procedure continues until obtain two values of 2^n bits, where n is the initial number of bits. Finally, the sum is done with CLA or RCA circuits.

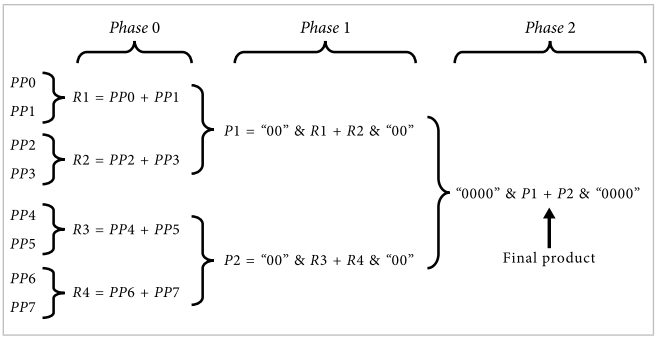

Figure 5. Addition of partial products for an array FP multiplier of 8 bits.

WALLACE TREE MULTIPLIER

The Wallace tree is an algorithm to implement fast multipliers. In this method the sum of partial products is carried out with an interconnection arrangement of adders to eliminate the problem of carry propagation.

In this method the multiplicand y is multiplied by the multiplier x, to generate the partial products. Then, they are added following the interconnection arrangement of carry-save adders (CSA) [17] to produce two rows of partial products. Finally, they are added with any high-speed adder. The Wallace tree requires compressors and full adders [3], [14].

Figure 6 shows the Wallace tree of an 8-bit multiplier without truncation. First, the partial products of multiplication are obtained. Then, as shown in Figure 6, the partial products are reduced to 2 rows through full adders and compressors. After that, the reduced partial products are grouped, which can be added with CLA or RCA circuits. For truncated multipliers, only the partial products of the n+1 bits of greater weight are generated. Then, the same algorithm of the Wallace tree is applied.

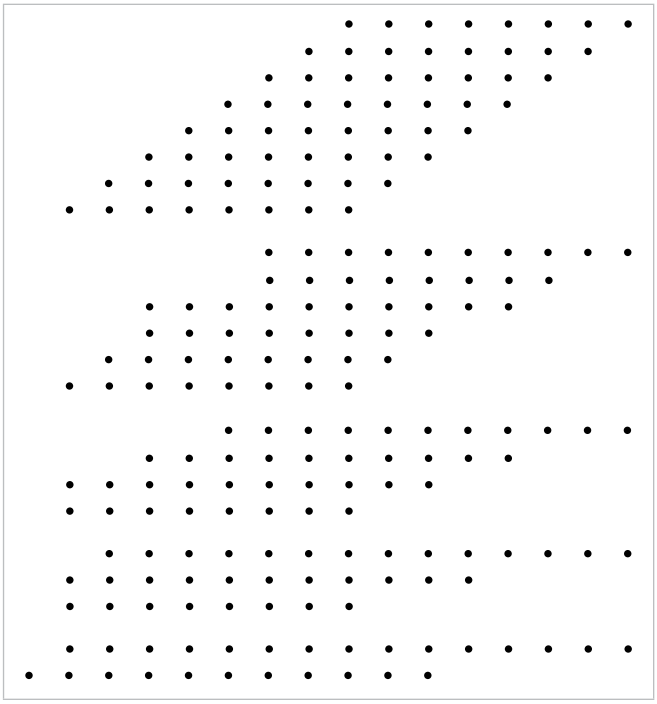

Figure 6. Wallace Tree FP multiplier of 8 bits (non-truncated).

BOOTH-2 MULTIPLIER

Booth's method multiplies two signed binary numbers in two's complement notation. The Booth algorithm is used to calculate the multiplication of signed integers. However, this method does not calculate partial products. It only uses displacement and adder circuits. It is based on the Booth-r coding [15]–[16].

Cultura Científica y Tecnológica • Vol. 18, Issue 1
Pages 1-9 • January-April 2021

Booth-r coding has a signed binary value in another format that can be interpreted as an equivalent of the original number. A signed binary value can be represented by

$$a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \ldots + a_0 \quad (2)$$

where $a_{n-1} \in \{-1,0\}$ and $a_i \in \{0,1\}$. Booth-r coding of a signed number a represented by (2) and $n = r \cdot m$ bits, is obtained by

$$b_0 = -a_{r-1}2^{r-1} + a_{r-2}2^{r-2} + \ldots + a_1 2 + a_0 \quad (3)$$

$$b_i = -a_{i\cdot r+r-1}2^{r-1} + a_{i\cdot r+r-2}2^{r-2} + \ldots + a_{i\cdot r+1}2 + a_{i\cdot r} + a_{i\cdot r-1} \quad (4)$$

where $i \in \{1, 2, \ldots, m-1\}$, a is the original binary signed value, b is the number encoded in Booth-r, n is the number of bits, r the degree of Booth and m the number of operands for an encoded element. The result (b_{m-1}, b_{m-2}, ..., b_0) will have components b_i in the range of $[-2^{r-1}, 2^{r-1}]$.

In this method, the adjacent pairs of bits of x, the multiplier, are examined. All the bits x_i and x_{i-1} are compared, with i increasing from 0 to $n-1$. Then based on the comparisons result, an action is performed on a register called product that will contain the result of the multiplication.

When x_i and x_{i-1} are the same, the product is not altered.

When $x_i = 0$ and $x_{i-1} = 1$, the multiplicand multiplied by 2^i is added to the product.

When $x_i = 1$ and $x_{i-1} = 0$, the multiplicand multiplied by 2^i is subtracted from the product.

Booth-r coding increases its complexity and the required hardware as its degree increases, but the number of operations to obtain a product is reduced.

In Booth-2, the relation $2m = n$ is established in Eq. (3) and (4). Therefore, the half of the elements is required to encode a number if it is compared to the number of bits of the binary number. The Booth-2 method has a small difference. Now three bits are compared (see Table III). Thus, the possible actions on the multiplier and the displacements to the right are increased to 2.

TABLE III
Possible Actions in Booth-2 Algorithm

Case	Action
000	No action
001	Add multiplicand × 1
010	Add multiplicand × 1
011	Add multiplicand × 2
100	Subtract multiplicand × 2
101	Subtract multiplicand × 1
110	Subtract multiplicand × 1
111	No action

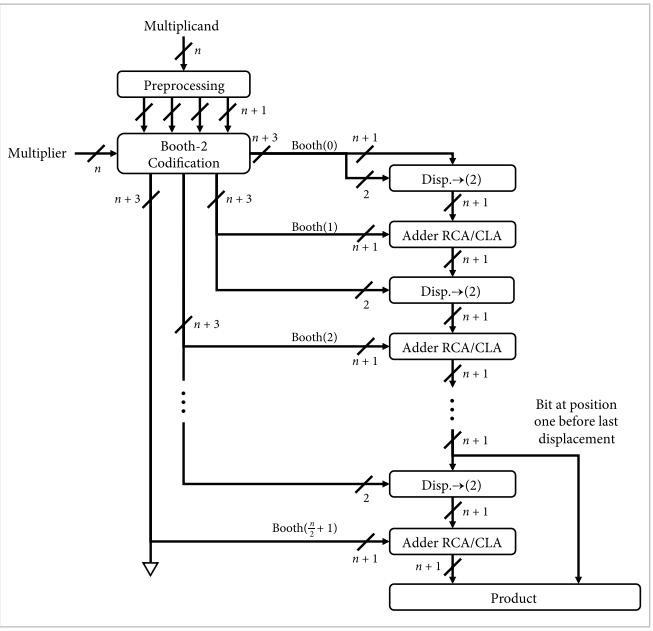

Figure 7. Booth-2 FP truncated multiplier of 8 bits.

It is important to mention that the sign bit in a floating-point multiplication under the IEEE-754:2008 standard is not necessary for the calculation of m. However, the Booth-2 multiplier requires it. On this case two bits are used to keep the product unchanged by the displacement of the multiplicand.

In Figure 7 the architecture without truncation of the Booth-2 multiplier is shown. In this multiplier, the multiplicand must be presented with a positive and negative sign and multiplied by 2 in both cases.

HARDWARE IMPLEMENTATION

The multipliers were designed in VHDL, VHSIC (Very High-Speed Integrated Circuit), and HDL (Hardware Description Language) [10]. Then a synthesis process

DOI: 10.20983/culcyt.2021.1.2.4

ISSN (e): 2007-0411

Cultura Científica y Tecnológica • Vol. 18, Issue 1
Pages 1-9 • January-April 2021 CULCYT

was carried out with the Alliance EDA (Electronic Design Automation) tools to obtain the hardware designs [18]. Finally, the designs were fabricated with the On Semiconductor C5 CMOS technology, which is characterized by a minimum channel length of 0.5 µm. This technology was sponsored by the MOSIS Inc. Educational Program.

LOGICAL AND PHYSICAL SYNTHESIS OF THE SYSTEM

The logical and physical synthesis process was carried out through the flowcharts of Figures 8 and 9. The synthesis was implemented using the Alliance EDA tools, which is a set of VLSI design tools and standard cell libraries that were developed in the Pierre et Marie Curie laboratory in Paris, France [18]. With these tools, a behavioral description is translated into a structural description. And finally, a transistor level layout is obtained.

The beginning of a design is the behavioral description of each module of the system in VHDL. Then the tools of the flowchart, shown in Figure 8a, are used to implement a logical synthesis. Subsequently, the physical synthesis is performed as shown in Figure 8b, which begins with a structural description. Then, as shown in Figure 9, the place and route process are carried out, placing the standard cells and interconnecting them. Likewise, the declaration of pads is made for the external signals, thus forming the complete IC layout.

First VASY (see Figure 8a) is used to verify the syntax and convert the high-level instructions to an understandable language by Alliance. After that, the behavior is simulated with ASIMUT through a pattern file that sets the input values. Next, BOOM is executed, which optimizes and converts VASY instructions into simple Boolean equations. Then, BOOG analyzes the equations to obtain the equivalent function, using the standard cells provided by the SXLIB library. Also, this tool generates the schematic system and could be simulated with ASIMUT. The structural modules, to form the complete system, are connected using GENLIB (see Figure 8b). Subsequently, LOON is used to optimize the critical path by introducing buffers and reducing capacitance. Once the interconnected structural system is obtained, OCP is used to place the standard cells and establish the physical inputs and outputs (see Figure 9). After that, the transistors of each cell are interconnected with NERO. Finally, the layout of the circuit is obtained. LVX generates a list of nodes to compare the layout with the structural file using COUGAR. The obtained layout has generic dimension units (λ), which can be scaled, allowing the designs to be fabricated with different VLSI CMOS technologies.

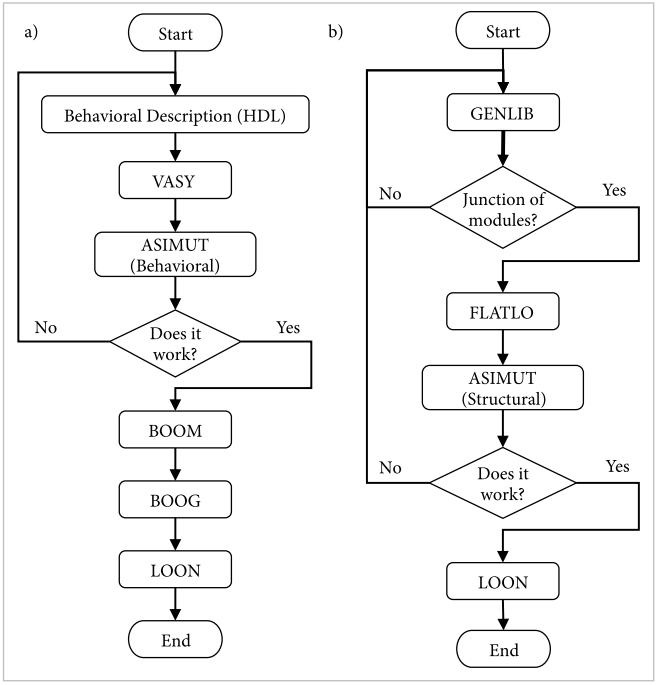

Figure 8. Flowchart for a) logic and b) physical synthesis, using Alliance EDA tools.

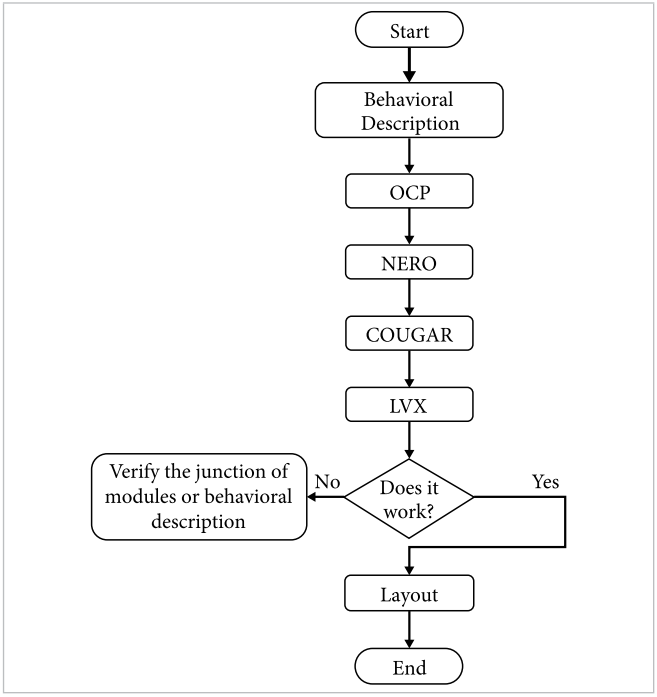

Figure 9. Flow chart for place and route and layout generation, using Alliance EDA tools.

DOI: 10.20983/culcyt.2021.1.2.4
ISSN (e): 2007-0411

III. RESULTS AND DISCUSSION

The results presented here, were obtained during the synthesis process depicted on Figures 8 and 9. The comparison of the area consumption (in generic units of λ^2) required by the FP multipliers is shown in Figure 10. The lowest area consumption is obtained with Wallace tree multipliers with truncation and RCA adders. Meanwhile, array multipliers without truncation and RCA adders consume more area. Table IV shows the transistors quantity obtained by COUGAR tool, of non-truncated/CLA versions of multipliers. These results agree with area consumption of Figure 10.

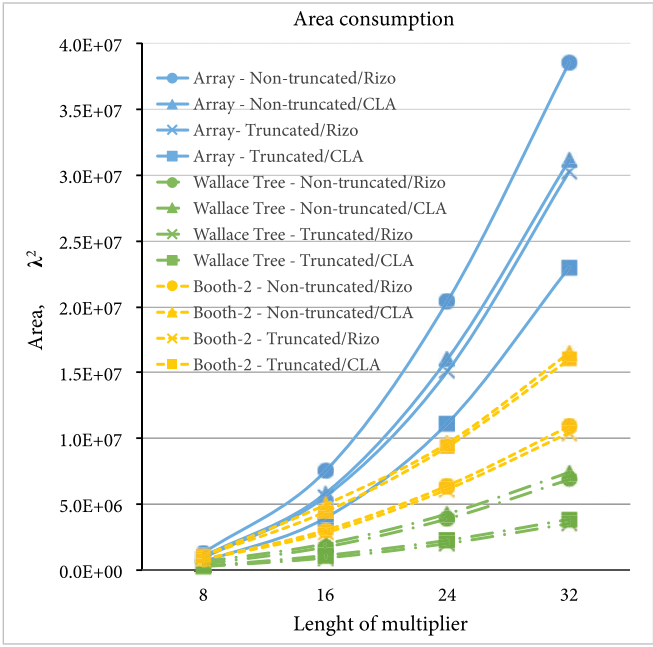

Figure 10. Area consumption in FP multipliers.

The processing time (or delay) of the FP multipliers designed are compared in Figure 11. As can be seen the Wallace tree multiplier truncated with CLA adders spent the least of times. The critical path delay of Wallace tree multipliers is proportional to the logarithm of the number of bits [3], [14]. The Wallace tree multiplier can be implemented only in signed integers and it is avoided in low power applications because its wiring excess increases the power consumption. On the other hand, the Booth-2 multiplier truncated with RCA adders presented the longest delay. It is not possible to determine which algorithm is more efficient in terms of speed. However, it can be seen that the higher the area consumption, the higher the operating speed.

TABLE IV
QUANTITY OF TRANSISTORS FOR NON-TRUNCATED / CLA MULTIPLIERS

NUMBER OF BITS	WALLACE TREE	BOOTH-2	ARRAY
8	2404	2980	8610
16	10572	12152	35558
24	22868	26198	96118
32	40178	43548	190290

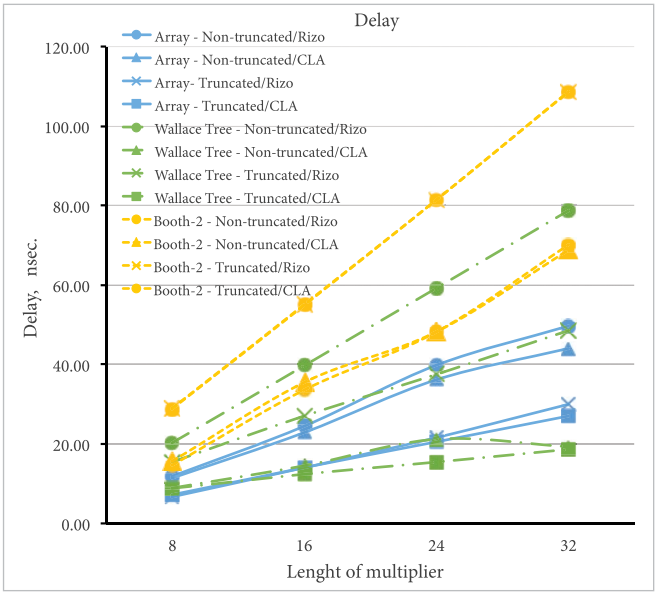

Figure 11. Processing time in FP multipliers.

As shown in Figure 12, the floating-point multiplier with a Wallace tree architecture required less area for its fabrication, while the array multiplier required the largest area. Also, the processing time of floating-point multipliers is shown in Figure 12. The Wallace tree architecture presented the shortest delay and Booth-2 architecture the longest. These results are important because as can be seen the critical part in a floating-point multiplier design is the FP portion.

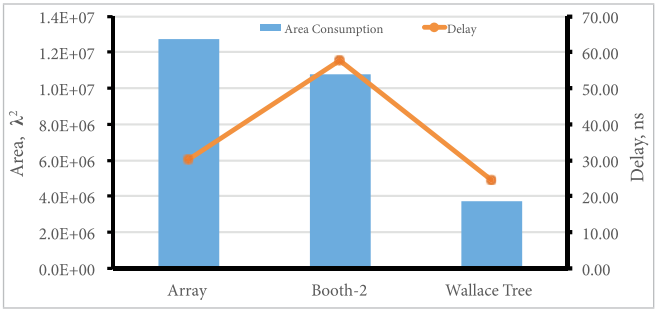

Figure 12. Area consumption and delay in floating-point multipliers.

Cultura Científica y Tecnológica • Vol. 18, Issue 1
Pages 1-9 • January-April 2021 CULCYT

In the design of floating-point multipliers, we can see that if the area consumption is high, the system speed will be as well. However, this statement is not fulfilled in all cases. The Wallace tree floating-point multiplier presented both the lowest area consumption and the least delay. However, it is known that the power consumption is high [3], [14].

In this work, parameter *m* was truncated to reduce the area consumption, introducing a precision error in the multiplication result. One solution is to use Booth-2 floating-point multipliers with truncation. They exhibit a null truncation error since the lower weight bits are calculated and they influence those of higher weight but are subsequently discarded. Also, as can be seen in Figure 12 its parameters are placed in the limits of area and processing time.

Therefore, they can be used in applications that require more precision, over speed or area consumption. In addition, Booth-2 multiplier reduces its area consumption compared to the other truncated architectures that introduce a precision error. Two ICs corresponding to the Both-2 and Wallace tree FP multipliers were successfully fabricated with a 0.5-µm CMOS technology. In Figure 13a is shown the final layout of a Wallace tree FP multiplier of 8 bits. The printed circuit board (PCB) for the testing of ICs is shown in Figure 13b. The physical dimensions of the design, without considering the pads, were 510 × 528.6 µm for the Wallace tree multiplier, and 690 × 670 µm for the Both-2 multiplier. The physical dimensions obtained validate the results previously discussed in Figure 10.

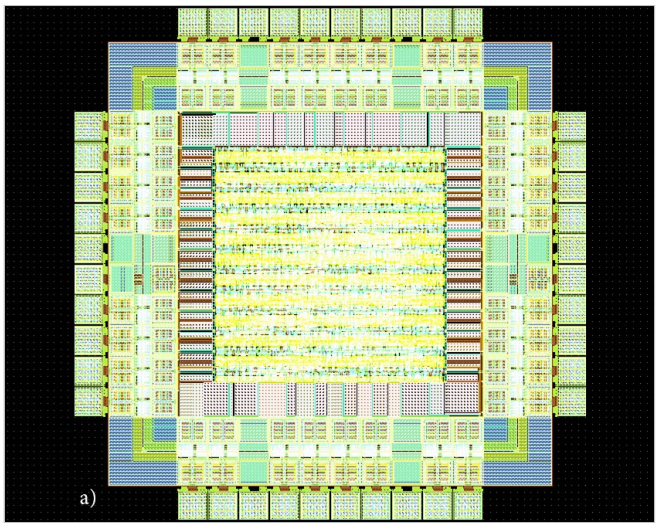

Figure 13a. Integrated circuit of 8 bits Wallace tree FP multiplier. Final layout.

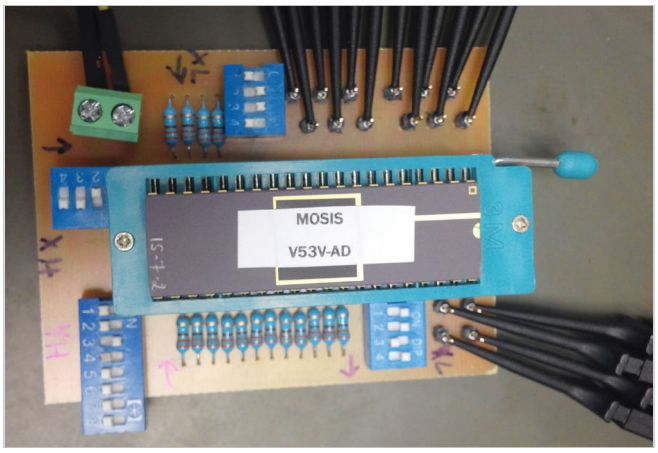

Figure 13b. Integrated circuit of 8 bits Wallace tree FP multiplier. Testing PCB.

IV. CONCLUSIONS

Three different fixed-point multipliers (Wallace Tree, Array and Booth-2) were successfully designed in VHDL. The multipliers were compared and analyzed based on area and delay parameters. The lowest area consumption is obtained with Wallace tree multipliers with truncation and RCA adders. Meanwhile, array multipliers without truncation and RCA adders consume more area. Wallace tree fixed-point multiplier with truncation and CLA adders presented the least delay. Meanwhile, the Booth-2 fixed-point multiplier with truncation and RCA adders presented the longest delay. In floating-point versions, the Booth-2 multiplier calculated the mantissa without truncation error and the required area was reduced. By using free EDA tools, it was possible to design and fabricate two integrated circuits in a 0.5-µm CMOS technology successfully. These integrated circuits correspond to the Wallace tree and Both-2 fixed-point multipliers of 8 bits, which were tested to corroborate their correct operation.

REFERENCES

[1] R. Shanmuganathan and K. Brindhadevi, "Comparative analysis of various types of multipliers for effective low power," *Microelectron. Eng.*, vol. 214, pp. 28-37, 2019, doi: 10.1016/j.mee.2019.04.015.

[2] V. Leon, S. Xydis, D. Soudris, and K. Pekmestzi, "Energy-efficient VLSI implementation of multipliers with double LSB operands," *IET Circuits, Devices Syst.*, vol. 13, no. 6, pp. 816-821, 2019, doi: 10.1049/iet-cds.2018.5039.

[3] I. Hussain and M. Kumar, "A Fast and Reduced Complexity Wallace Tree Multiplier," *Journal of Active and Passive Electronic Devices*, vol. 12, no. 1-2, pp. 63-71, 2017.

[4] P. Lokesh, U. Somalatha, and S. Chandana, "VLSI Modeling of high performance aging aware multiplier by using adaptive hold logic circuit," *International Journal of Engineering Research and Applications*, vol. 8, no. 2, pp. 7-12, 2018.

[5] M. Jhamb, Garima, and H. Lohani, "Design, implementation and performance comparison of multiplier topologies in power-delay space," *Eng. Sci. Technol. an Int. J.*, vol. 19, no. 1, pp. 355-363, 2016, doi: 10.1016/j.jestch.2015.08.006.

[6] A. Kamaraj and P. Marichamy, "Design of integrated reversible fault-tolerant arithmetic and logic unit," *Microprocess Microsyst*, vol. 69, pp. 16-23, 2019, doi: 10.1016/j.micpro.2019.05.009.

[7] M. Ito, D. Chinnery, and K. Keutzer, "Low power multiplication algorithm for switching activity reduction through operand decomposition," *Proceedings 21st International Conference on Computer Design*, 2003, pp. 21-26, doi: 10.1109/ICCD.2003.1240868.

[8] Y. Jiang, A. Al-Sheraidah, Y. Wang, E. Sha, and J.-G. Chung, "A novel multiplexer-based low-power full adder," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 51, no. 7, pp. 345-348, July 2004, doi: 10.1109/TCSII.2004.831429.

[9] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, Boston: Addison-Wesley, 2011.

[10] *1076-2008 - IEEE Standard VHDL Language Reference Manual*, IEEE Standards Association (IEEE SA), USA, 2009.

[11] *754-2008 - IEEE Standard for Floating-Point Arithmetic*, IEEE Standards Association (IEEE SA), New York, 2008.

[12] M. Gök, "A novel IEEE rounding algorithm for high-speed floating-point multipliers," *Integration*, vol. 40, no. 4, pp. 549-560, 2007, doi: 10.1016/j.vlsi.2006.12.001.

[13] Z. Huang and M. D. Ercegovac, "High-performance low-power left-to-right array multiplier design," in *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 272-283, March 2005, doi: 10.1109/TC.2005.51.

[14] C. S. Wallace, "A Suggestion for a Fast Multiplier," in IEEE Transactions on Electronic Computers, vol. EC-13, no. 1, pp. 14-17, Feb. 1964, doi: 10.1109/PGEC.1964.263830.

[15] A. D. Booth, "A Signed Binary Multiplication Technique," *Q J Mech Appl Math*, vol. 4, no. 2, pp. 236-240, 1951, doi: https://doi.org/10.1093/qjmam/4.2.236.

[16] T.-A. Chu, "Booth multiplier with low power high performance input circuitry," U.S. Patent 6,393,454 B1, May. 21, 2002.

[17] J. G. Earle, "Latched carry save adder circuit for multipliers," U.S. Patent 3,340,388, Sept. 5, 1967.

[18] "Alliance/Coriolis VLSI CAD Tools." Coriolis.lip6.fr. http://coriolis.lip6.fr/ (accessed March, 15, 2021).