An in-depth-examination: comparative analysis of multiplication hardware accelerator algorithms in VHDL for 8-Bit Systems (WTM), (PBM) and (BWM) synthesized on an ALTERA-CYCLONE-II-DE1-Board

Examinación a profundidad: análisis comparativo de algoritmos aceleradores de hardware de multiplicación en VHDL para sistemas de 8 bits (WTM), (PBM) y (BWM) sintetizados en una placa ALTERA CYCLONE II DE1

Exame aprofundado: Análise comparativa de algoritmos de aceleração de hardware de multiplicação VHDL para sistemas de 8 bits (WTM), (PBM) e (BWM) sintetizados em uma placa ALTERA CYCLONE II DE1

Andres Gonzalo Hernandez Ortega[1]
Braian Stiven Avella Rivera[2]
Oscar Fernando Vera[3]
Jorge Orlando Bareño Quintero[4]

Received: January 25th, 2024
Accepted: March 30th, 2024
Available: May 7th, 2024

How to cite this article:
A.G. Hernandez Ortega, B.S. Avella Rivera, O.F. Vera, J.O. Bareño Quintero, "An In-Depth-Examination: Comparative Analysis of Multiplication Hardware Accelerator Algorithms in VHDL for 8-Bit Systems (WTM), (PBM) and (BWM) synthesized on an ALTERA-CYCLONE-II-DE1-Board," *Revista Ingeniería Solidaria,* vol. 20, no. 2, 2024.
doi: https://doi.org/10.16925/2357-6014.2024.02.10

Research article. https://doi.org/10.16925/2357-6014.2024.02.10

[1] Email: andresgonzalo.hernandez@uptc.edu.co
ORCID: https://orcid.org/0000-0002-3190-6660
CVLAC: https://scienti.minciencias.gov.co/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0000098693
Maestría en Ingeniería - Universidad Pedagogica y Tecnologica de Colombia UPTC.
Universidad nacional abierta y a Distancia UNAD.

[2] Email: braian.avella@uptc.edu.co
ORCID: https://orcid.org/0000-0002-3035-7460
CVLAC: https://scienti.minciencias.gov.co/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0001632169
Maestría en Ingeniería - Universidad Pedagogica y Tecnologica de Colombia UPTC

[3] Email: oscar.vera@uptc.edu.co
ORCID: https://orcid.org/0000-0002-0008-8343
CVLAC: https://scienti.minciencias.gov.co/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0000372340

[4] Email: jorge.bareno@unad.edu.co
ORCID: https://orcid.org/0000-0002-4821-8194
CVLAC: https://scienti.minciencias.gov.co/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0001339796
Universidad nacional abierta y a Distancia UNAD

Abstract

This article presents an implementation and comparative analysis of 8-bit Wallace Tree Multiplier (WTM), Parallel Booth Multiplier (PBM), and Baugh-Wooley Multiplier (BWM) algorithms.

Introduction: this article results from research conducted for a Master's degree in Engineering at the Pedagogical and Technological University of Colombia (UPTC) between 2022 and 2024. It analyzes and compares the performance of three multiplication algorithms (WTM, PBM, and BWM), focusing on variables such as operation time and the number of logical elements used.

Problem: computing has advanced rapidly, enabling multiple operations on a single chip. This has increased the demand for components that execute tasks quickly while occupying minimal space. Multipliers are crucial in applications such as filters, DSP circuits, and fast Fourier transforms.

Objective: to analyze and compare the performance of three multiplication algorithms—WTM, PBM, and BWM—tailored for 8-bit systems.

Methodology: the research methodology was designed to ensure robustness and reliability. It began with formulating objectives and identifying key variables. Articles were selected based on their contributions to understanding multiplication algorithms and programming languages. The research included designing and comparing 8-bit multiplier algorithms (WTM, PBM, and BWM).

Results: an analysis of the results identified the variables that contributed to significant performance improvements for each algorithm.

Conclusions: the project successfully improved the efficiency of the algorithms by utilizing various register shifts and multipliers based on the operational case that benefited them the most. It achieved improvements in both efficiency and operation time concerning the use of logical elements.

Originality: this research formulates strategies for applying and comparing multiplication algorithms, differentiating data processing based on specific data characteristics.

Limitations:

- The lack of testing systems for the implementations.
- The study focused on comparing three multiplication algorithms, limiting generalizability.
- Performance metrics.

Keywords: Multipliers, VHDL, ModelSim, FPGA, Full Adders, Half Adders.

Resumen

Este artículo presenta una implementación y análisis comparativo de los algoritmos de multiplicación Wallace Tree Multiplier-(WTM), Parallel Boot Multiplier-(PBM) y Baugh-Wooley Multiplier-(BWM) de 8 bits.

Introducción: este artículo es el resultado de una investigación realizada para la Maestría en Ingeniería en la Universidad Pedagógica y Tecnológica de Colombia (UPTC) entre 2022 y 2024, los resultados se enfocan en variables como el tiempo de operación y el número de elementos lógicos utilizados.

Problema: la informática ha avanzado rápidamente, realizando múltiples operaciones en un solo chip. Esto ha incrementado la demanda de componentes que ejecuten tareas rápidamente y ocupen un espacio mínimo. Los multiplicadores son cruciales en aplicaciones como filtros, circuitos DSP y transformadas rápidas de Fourier.

Objetivo: analizar y comparar el rendimiento de tres algoritmos de multiplicación—Wallace Tree, Booth Parallel y Baugh-Wooley—adaptados para sistemas de 8 bits.

Metodología: la metodología comienza con la formulación de objetivos e identificación de variables clave como el tiempo de operación, el número de elementos lógicos. Se realizó una revisión bibliográfica exhaustiva y se seleccionaron artículos basados en su relevancia para el análisis del rendimiento de algoritmos de multiplicación y su aplicabilidad.

Resultados: un análisis de los resultados identificó las variables que lograron mejoras significativas en el rendimien-

to de cada algoritmo.

Conclusiones: el proyecto logra mejorar la eficiencia de los algoritmos utilizando varios desplazamientos de registros y multiplicadores basados en el caso operativo que más los beneficie, el proyecto logró mejorar la eficiencia y tiempo de operación, En cuanto al uso de elementos lógicos.

Originalidad: esta investigación formula estrategias para aplicar y comparar algoritmos de multiplicación con diferenciación en el procesamiento de datos basándose en características específicas de los datos analizados.

Límites:

- *La falta de sistemas de prueba para las implementaciones.*
- *El estudio se centró en la comparación de tres algoritmos de multiplicación en sistemas de 8 bits en VHDL, limitando su generalización.*
- *Las métricas de rendimiento pueden no captar todas las dimensiones del rendimiento de los algoritmos y lenguajes de programación.*

Palabras clave: Multiplicadores,VHDL,MODELSIM,FPGA,Sumadores completos,Sumadores parciales.

Resumo

Este artigo apresenta uma implementação e análise comparativa dos algoritmos de multiplicação Wallace Tree Multiplier-(WTM), Parallel Boot Multiplier-(PBM) e Baugh-Wooley Multiplier-(BWM) de 8 bits.

Introdução: este artigo é resultado de uma pesquisa realizada para o Mestrado em Engenharia da Universidade Pedagógica e Tecnológica da Colômbia (UPTC) entre 2022 e 2024. Os resultados se concentram em variáveis como tempo de operação e número de elementos lógicos utilizados.

Problema: a computação avançou rapidamente, realizando diversas operações em um único chip. Isso aumentou a demanda por componentes que executem tarefas rapidamente e ocupem o mínimo de espaço. Multiplicadores são cruciais em aplicações como filtros, circuitos DSP e transformadas rápidas de Fourier.

Mirar: analise e compare o desempenho de três algoritmos de multiplicação — Wallace Tree, Booth Parallel e Baugh-Wooley — adaptados para sistemas de 8 bits.

Metodología: a metodologia começa com a formulação de objetivos e a identificação de variáveis-chave, como o tempo de operação e o número de elementos lógicos. Foi realizada uma revisão abrangente da literatura e os artigos foram selecionados com base em sua relevância para a análise do desempenho do algoritmo de multiplicação e sua aplicabilidade.

Resultados: uma análise dos resultados identificou as variáveis que alcançaram melhorias significativas no desempenho de cada algoritmo.

Conclusões: o projeto alcança eficiência de algoritmo aprimorada usando vários deslocamentos de registro e multiplicadores com base no caso operacional que mais os beneficia, o projeto alcançou eficiência e tempo de operação aprimorados, em termos de uso de elementos lógicos.

Originalidade: esta pesquisa formula estratégias para aplicar e comparar algoritmos de multiplicação e diferenciação no processamento de dados com base em características específicas dos dados que estão sendo analisados.

Limites:

- *Falta de sistemas de testes para implementações.*
- *O estudo se concentrou na comparação de três algoritmos de multiplicação em sistemas de 8 bits em VHDL, limitando sua generalização.*
- *As métricas de desempenho podem não capturar todas as dimensões do desempenho de algoritmos e linguagens de programação.*

Palavras-chave: Multiplicadores, VHDL, MODELSIM, FPGA, Somadores completos, Somadores parciais.

1. INTRODUCTION

In the past few decades, there has been rapid progress in the field of computing, enabling the execution of multiple operations on a single chip. Among these operations, multiplication plays a pivotal role in arithmetic computations. The demand for high-speed multipliers has remained consistently high over time [1-2]. Among various components, multipliers hold significant importance in applications such as filters, DSP circuits, machine learning, and fast Fourier transforms [3-7].

The advancement of circuits in the current digital era is constrained by the exploration of alternative nanodevices, moving beyond sole reliance on CMOS technology [8-10]. Traditional multipliers consume substantial energy and exhibit limited execution speed, creating a growing necessity for high-performance multipliers [11-12].

There are several types of multipliers, each designed to perform multiplication operations, although they differ in their method of obtaining partial products [13]. This article explores some of these multiplier types, as they vary in their approach and functionality, employing cascaded carry adders composed of half-adder and full-adder operators.

These multipliers are implemented through various programming routines. The Wallace algorithm was developed by computer scientist Chris Wallace in 1964. The Baugh-Wooley multiplier algorithm was created between 1973 and 1979, while the Booth multiplier algorithm was invented by Andrew Donald Booth in 1950 [14]. These methods remain among the most efficient for performing intensive multiplication operations with low hardware costs, even today [15-16]. They play a crucial role in applications such as trigonometric transformations [17-18], Fast Fourier Transform (FFT) [3,17], neural network implementations [17,19], and any process that requires multiplication operations.

The development was carried out on a Field-Programmable Gate Array (FPGA) platform. FPGAs are widely used due to their high efficiency in performing mathematical operations with low energy consumption [17]. These characteristics have captured the interest of the scientific community, which seeks to harness the inherent parallelism of these algorithms for efficient implementation across various software and hardware platforms [17,19].

The design of multipliers typically involves three steps:

Generation of Partial Products: This involves generating the intermediate products resulting from multiplying individual bits of the two input numbers.

Compression Tree Design: The partial products generated in the previous step need to be added together in a structured manner. This requires designing a

compression tree that efficiently combines these partial products to obtain a reduced set of terms.

Final Adder Design: The reduced set of terms from the compression tree is then summed using a final adder to produce the final product of the multiplication.

For FPGA implementations, these steps present distinct design challenges, as there are numerous potential FPGA assignments for each stage. Therefore, each step requires its own combinatorial optimization [20].

Advanced digital design is achieved using Hardware Description Language (HDL), specifically VHDL (VHSIC Hardware Description Language) [18]. This process is utilized to create complex digital systems such as processors, embedded systems, communication systems, and other highly sophisticated digital architectures. High-speed multiplier designs are essential in digital arithmetic, each with its own advantages and disadvantages. The selection of a specific algorithm depends on the project's requirements [3,14,21].

This work presents a VHDL implementation of three 8-bit multiplier algorithms: Wallace Tree Multiplier (WTM), Parallel Booth Multiplier (PBM), and Baugh-Wooley Multiplier (BWM) on an FPGA platform. This implementation allows for the comparison of operation modes and performance. The proposed architecture provides an open-source VHDL implementation that meets the requirements of most applications utilizing this module.

The remainder of the document is structured as follows: The Experimental Development section revisits the underlying concepts of the algorithms and the proposed architecture for their implementation. The Results section outlines the procedures performed to verify the developed architecture and presents the obtained outcomes. Finally, the Conclusions section summarizes the work.

The Wallace Multiplier (WTM):

The Wallace multiplier is a binary multiplication method that utilizes a tree-like structure of addition and shifting to efficiently perform multiplication. This approach is widely used in high-speed and low-power digital circuit implementations. In January 2022, a scientific study was published comparing two Wallace multiplier architectures implemented using transistors [22].

Parallel Booth Multiplier (PBM):

The Booth algorithm, used in modular arithmetic, enables more efficient multiplication of large numbers. This algorithm is well-suited for large-number multiplication in distributed or parallel systems. In Booth multiplication, partial product generation is optimized using a specific encoding technique, which can be illustrated with a flowchart. The multiplicand (BR) bits are grouped from left to right, and corresponding operations on the multiplier (QR) are performed to generate partial products. These partial products are then summed using a Carry-Save Adder (CSA) [23-25].

An approximate radix-8 Booth multiplier further reduces the number of addition steps in the multiplication process [26]. The design of the Parallel Booth Multiplier (PBM) employs the Barrett reduction technique to minimize intermediate products during multiplication. This reduces the number of operations needed to compute the final result. Additionally, the PBM algorithm can be easily parallelized, making it suitable for distributed systems and parallel processing environments.

The Baugh-Wooley Multiplier (BWM):

The BWM algorithm is based on the partial sum technique, which reduces the number of operations required to multiply two binary numbers. Instead of directly multiplying the two numbers, they are decomposed into smaller parts, which are then multiplied separately [4]. The partial results are subsequently summed to obtain the final product.

Reversible logic designs have been proposed for the Baugh-Wooley multiplier, but these designs have led to increased power consumption and substantial energy dissipation due to information loss in conventional approaches [27-28].

Multiplication and division are essential operations in communication systems. Traditionally, multipliers occupy larger areas, consume more power, and introduce significant latency [29]. These characteristics depend on the specific multiplication techniques employed. Therefore, implementing an efficient multiplier that minimizes both latency and power consumption is crucial.

The Baugh-Wooley multiplier has been used as a type of signed multiplier that incorporates both exact and approximate compressors to reduce the number of partial products generated. Exact compressors are used to compute the most significant bits, while approximate compressors generate the least significant bits. This approach reduces hardware requirements while maintaining high accuracy [30].

e-ISSN 2357-6014 / Vol. 20, no. 2 / may-august 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

2. MATERIALS AND METHODS

Figure 1 illustrates the input and output components of the Wallace multiplier, which has two 8-bit standard logic vector inputs and a 16-bit standard logic vector output.

Wallace Multiplier

```
--    Wallace Tree multiplier
--              _______
--  A(7-0)->|         |
--  B(7-0)->|  WTM    |->X(15-0)
--          |_________|
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity wallace8 is
    Port (  A : in  STD_LOGIC_VECTOR (7 downto 0);
            B : in  STD_LOGIC_VECTOR (7 downto 0);
            X : out  STD_LOGIC_VECTOR (15 downto 0));
end wallace8;
```

Figure 1. Code header and entity ports.
Source: Own work.

In the realm of arithmetic circuits, the Wallace tree emerges as a pivotal structure, characterized by its tree-like configuration of addition and shifting operations. Its primary function is to aggregate the partial products generated from the multiplication of individual bits within operands [30].

Constructed using a series of addition and shifting blocks, the Wallace tree enhances efficiency by streamlining the process of combining partial products. This efficiency is achieved by strategically reducing the number of addition operations required and minimizing overall shifting overhead [31]. As a result, the Wallace tree serves as a cornerstone in arithmetic circuit design, offering optimized solutions for complex multiplication tasks.

Figure 2 illustrates the structure of partial products and their distinct groupings, highlighting the combinations processed through full adders or half adders. The analysis reveals that implementing an 8-bit Wallace tree requires five stages. Additionally, it is noted that an OR operation is applied to the bits C68 and R68.

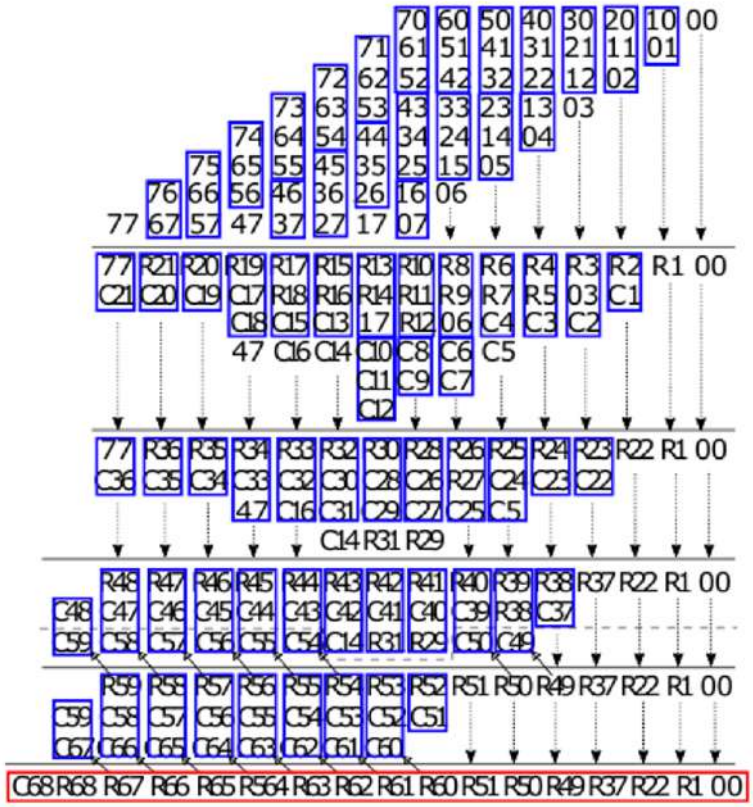

Figure 2. Aggregation of partial products to form the Wallace tree, five (5) stages.
Source: Own work.

The algorithm's design rationale involves individually implementing each component to accommodate the irregular access pattern inherent in the matrix of results and partial products. This approach is necessary due to the algorithm's unique requirements, which deviate from conventional architectures. Full adders and half adders are interconnected according to the stages outlined in Figure 2, ensuring proper alignment with the algorithm's processing flow. This meticulous component integration strategy optimizes execution efficiency and enables seamless handling of complex multiplication tasks.

Figure 1 illustrates that the Wallace multiplier has three ports: two 8-bit inputs (A and B) and a 16-bit output (X).

Figure 3 depicts the input and output components of the Parallel Booth Multiplier, which consists of two 8-bit standard logic vector inputs and one 16-bit standard logic vector output.

3) Parallel Booth Multiplier (PBM):

```
----------------------------------------------------------
--------     Parallel Booth Multiplier 8 bits     ----------
--------            State machine                 ----------
----------------------------------------------------------
--                        _______
--              A(7-0)->|       |
--              B(7-0)->|  PBM  |->X(15-0)
--                      |_______|
----------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity boothMult is
    generic (   nBits : integer := 8);
    port     (  A : in  std_logic_vector(nBits-1 downto 0);--M
                B : in  std_logic_vector(nBits-1 downto 0);--M
                X : out std_logic_vector(2*(nBits)-1 downto 0)
             );
end entity boothMult;
```

Figure 3. Block diagram and corresponding input-output ports for the Parallel Booth multiplier.
Source: Own work.

The design of the proposed multiplier is carried out through a sequence of steps: in the first stage, partial products are generated; in the second stage, recoded partial products are added until only two rows remain; and in the final stage, the corresponding rows are summed to obtain the final result. In this article, multiplication is implemented using the Booth Encoding technique, which relies mainly on repeated applications of multiplication and addition. The speed of these operations determines the execution speed and realization of the entire calculation. Since the multiplier typically requires the longest delay among the basic operational blocks in a digital system, the critical path is generally determined by the multiplier.

The VHDL encoding of the Parallel Booth Multiplier was performed using various components, including an encoder, as shown in Figure 4, which determines the appropriate step to be taken with the data frame. A full adder is used for bit-by-bit additions, and a multiplexer selection unit is also included. In Figure 5, the block diagram and ports of the Parallel Booth Multiplier are shown. Similar to the previous multipliers, it has two 8-bit inputs and one 16-bit output.

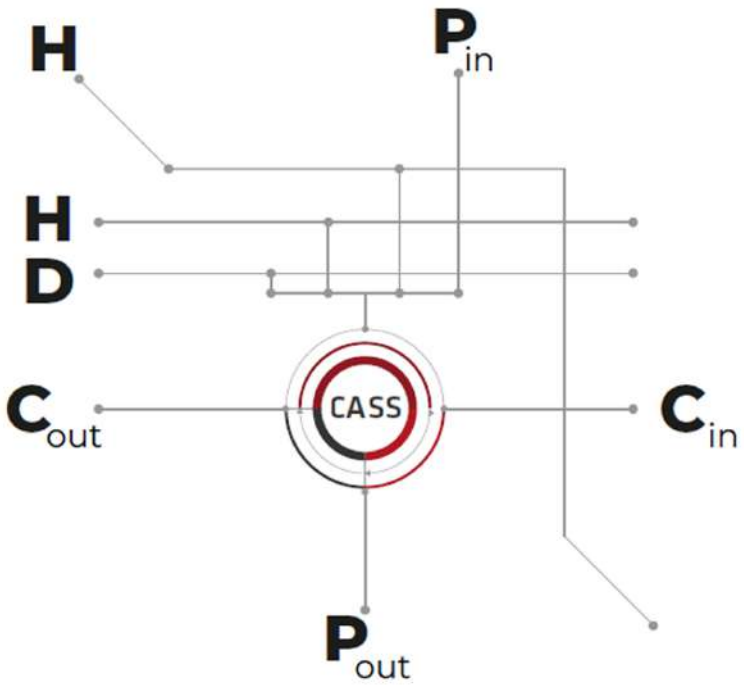

Figure 4. Controlled Add-Subtract-Shift Cell (CASS)
Source: Own work.

$$P_{out} = P_{\$n} \oplus x.H \oplus C_{\$n}.H$$
$$C_{out} = P_{\$n} \oplus D.x + C_{\$n} + x.C_{\$n}$$

H and D are control signals that denote the operation type to be executed:

If H=0, a non-arithmetic operation is indicated, resulting in the preservation of the previous partial product, where Pout equals Pin.

If H=1, an arithmetic operation is specified, necessitating the generation of a new Pout. The nature of the arithmetic operation is determined by D:

If D=0, the multiplicand bit, represented as x, is combined with Pin, using Cin as the carry- in sourced from the neighboring cell on the left.

If D=1, the multiplicand bit x is subtracted from Pin, employing Cin as the borrow-in and Cout as the borrow-out.

The signals H and D derive from the yi and yi-1 bits, following the principles of the Booth algorithm.

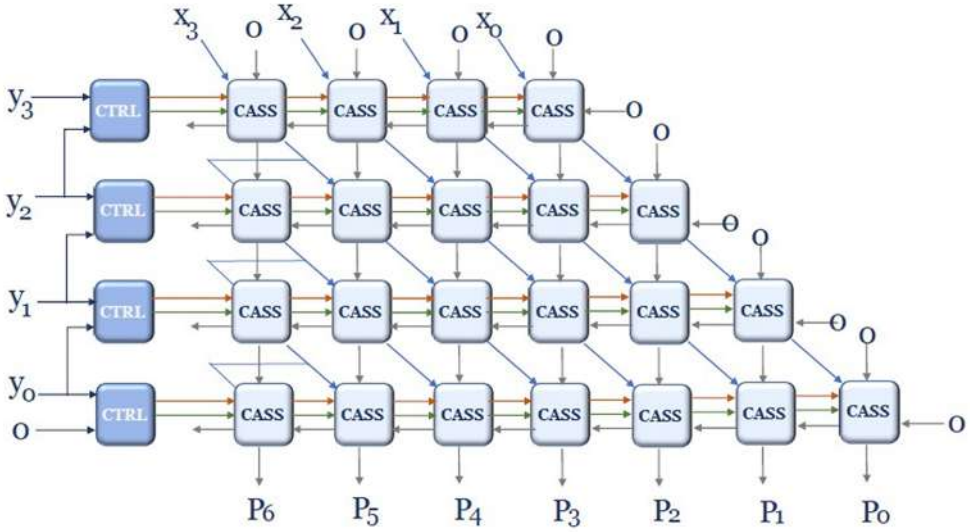

Figure 5. Multiplier Configuration Matrix
Source: Own work.

Baught-Wooley Multiplier (BWM):

In Figure 6, the input and output components of the Baugh Wooley Multiplier with 2 inputs of 8-bit standard logic vector type and one output of 16-bit standard logic vector type were shown.

```
----------------------------------------------------
-------    Multiplicador de Baugh Wooley de 8 bits
----------------------------------------------------
--                      _______
--                     |       |
--              A(7-0)->| Baugh |->S(15-0)
--              B(7-0)->| Wooley |
--             |        |_______|
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BaughWooley8 is
    generic(constant Len: natural := 8);
    port (
        A: in  std_logic_vector (Len-1 downto 0);--M
        B: in  std_logic_vector (Len-1 downto 0);--M
        X: out std_logic_vector (2*Len-1 downto 0)--
    );
end BaughWooley8;
```

Figure 6. Block diagram and corresponding input-output ports for the Baugh-Wooley multiplier.
Source: Own work.

The architectural blueprint of the Baugh-Wooley multiplier, as outlined in Figure 7, serves as the foundational framework for its structure. In this document, the conceptualization illustrated in Figure 8 for 4-bit systems was extrapolated to accommodate 8-bit configurations. However, upon examining the source code, it becomes clear that the initial phase involves creating a matrix primarily composed of AND gates, followed by the use of a generic matrix of full adders. This implementation highlights a strategic departure from the depicted schematic, underscoring the flexibility and adaptability of the Baugh-Wooley multiplier design methodology [32-33].

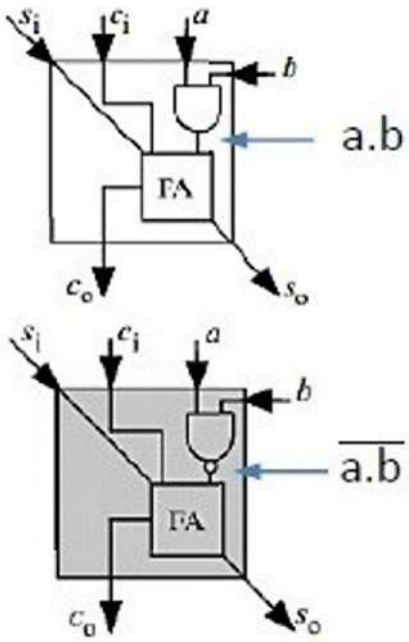

Figure 7. content of the cells that make up the 8-bit Baugh-Wooley multiplier consists.
Source: Own work.

In the Baugh-Wooley multiplier architecture, as depicted in Figure 8, the fundamental concept involves constructing a matrix of full adders. Within this matrix, one input is consistently set to '0' or the sum from the previous stage, while the other input receives the carry from the immediately preceding full adder. Notably, for the elements positioned at the edges, a NAND gate is used instead of the typical AND gate, as shown in Figure 7. Additionally, for the element located at position (n-1) x (n-1) — the bottom-left corner of the matrix — an AND gate is employed. It is important to note that this conceptual framework can be extended to accommodate multipliers with higher bit configurations, facilitating versatile multiplication operations.

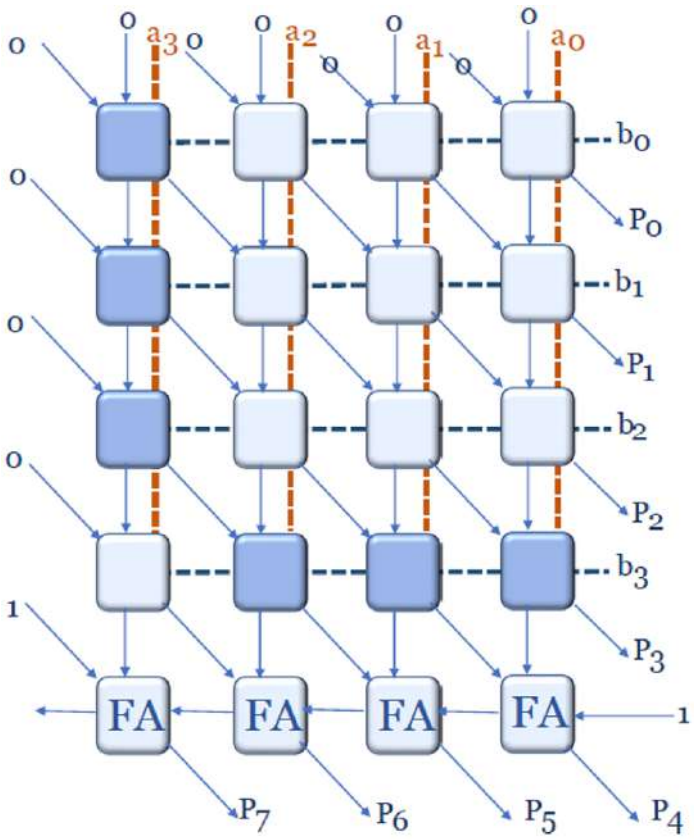

Figure 8. Block diagram of the 8-bit Baugh-Wooley multiplier.
Source: Own work.

In synthesizing the final row of full adders, as illustrated in Figure 8, an 8-bit Ripple Carry Adder (RCA) was integrated. This choice was made due to its efficient carry propagation across all bits, ensuring seamless addition of partial products generated during multiplication. Essential code for both the 8-bit RCA and full adder components was implemented to enable the multiplier's functionality. The output X of the 8-bit multiplier was then meticulously mapped, establishing the crucial correspondence between inputs and outputs. This careful design process highlights the robustness and efficiency of the 8-bit multiplier implementation in VHDL.

3. RESULTS

The Wallace Tree Multiplier (WTM) is an algorithm that utilizes the technique of parallel reduction to minimize the number of necessary additions and multiplications. It is particularly suited for multiplying large numbers due to its high efficiency in terms of

speed and resource usage. However, the complexity of its implementation increases with the size of the numbers, and specialized hardware is required to implement the algorithm.

To verify the proper functioning of the 8-bit Wallace Tree Multiplier, a simulation was performed in ModelSim. The testbench code is provided at the end of the document. Figure 9 displays the simulation results, with the graph plotted using a postscript plotter. After conducting the multiplications, the results were found to align with those presented in Figures 10 and 11.

Figure 9. Testbench using ModelSim of the 8-bit Wallace Tree Multiplier.
Source: Own work.

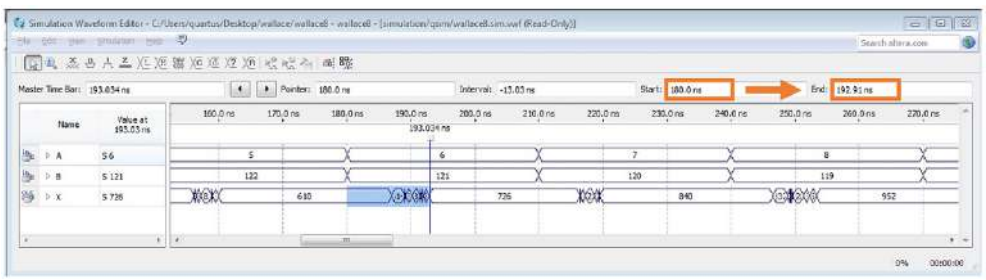

Figure 10. Time trial for the execution of sample operation.
Source: Own work.

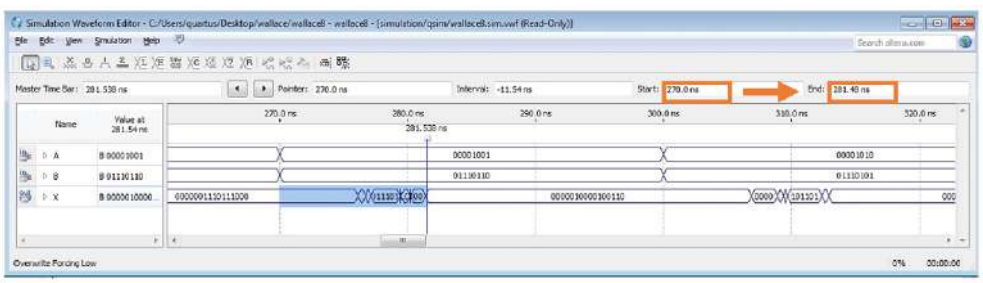

Figure 11. Time trial for performing a sample operation.
Source: Own work.

PARALLEL BOOTH MULTIPLIER

The Parallel Booth Multiplier (PBM) is an algorithm that employs the Barrett reduction technique to minimize intermediate products during multiplication [34-35]. Additionally, it can be easily parallelized, making it suitable for distributed systems and parallel processing setups. While it is more efficient than the Sequential Booth Multiplier (SBM), it is less efficient than the Wallace Tree Multiplier (WTM) in terms of speed.

As shown in Figures 12–14, the operation test benches illustrate the system's response to each operation in relation to time for the hardware accelerator algorithm of the Parallel Booth Multiplier.

Figure 12. Simulation using ModelSim of the 8-bit Parallel Booth Multiplier.
Source: Own work.

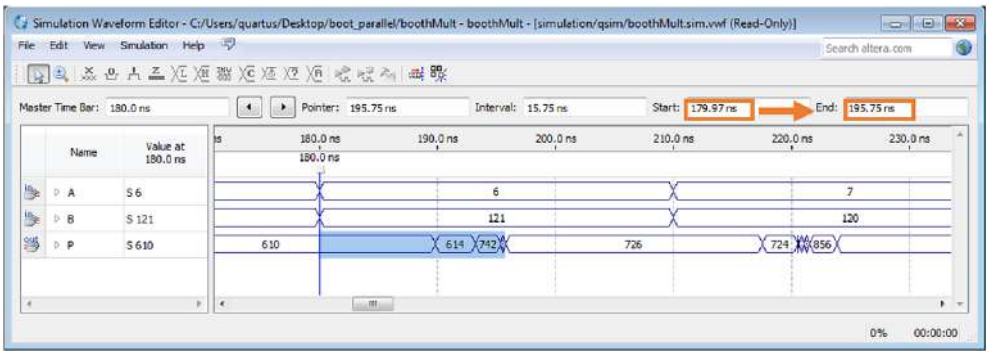

Figure 13. Time trial for the execution of sample booth operation.
Source: Own work.

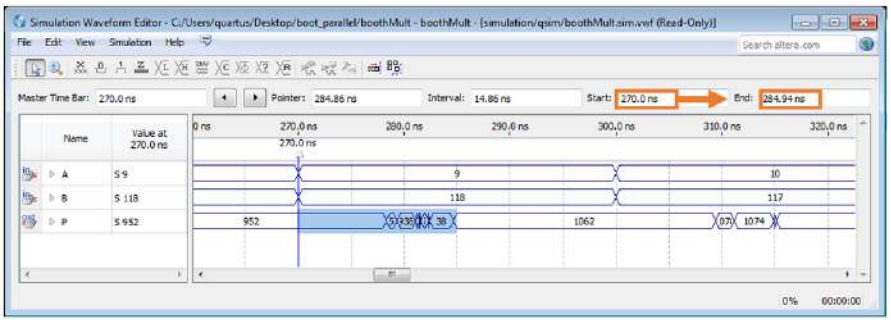

Figure 14. Time trial for performing a sample booth operation.
Source: Own work.

BAUGH WOLLEY MULTIPLIER

The Baugh-Wooley Multiplier (BWM) employs the technique of partial sums in a tree structure to reduce the number of required operations. It is particularly suited for multiplying large binary numbers, as it involves fewer operations compared to other multiplication algorithms. Its tree structure also facilitates easy implementation in parallel processing systems [36]. However, its efficiency decreases when multiplying numbers with an unequal count of ones and zeros.

As shown in Figures 15–17, the operation test benches illustrate the system's response to each operation in relation to time for the hardware accelerator algorithm of the Baugh-Wooley Multiplier.

Figure 15. Test bench using ModelSim of the 8-bit Baugh-Wooley Multiplier.
Source: Own work.

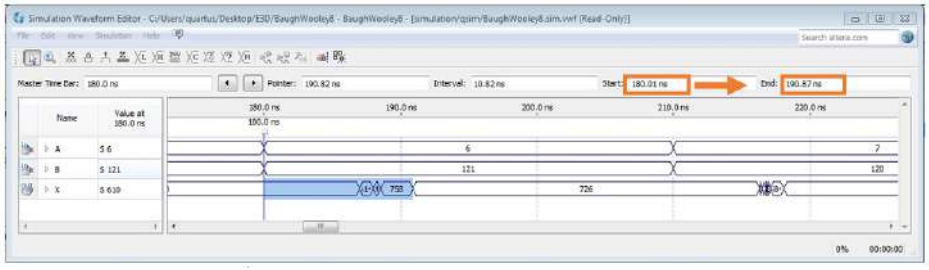

Figure 16. Time trial for the execution of sample baugh-wooley operation.
Source: Own work.

Figure 17. Time trial for the execution of sample baugh-wooley operation.
Source: Own work.

In the comparison of multiplication times, the efficiency of each method is clearly observed. The results show that the Baugh-Wooley Multiplier achieves outstanding performance in terms of speed at 10.82 ns, significantly surpassing the Wallace Multiplier at 13.03 ns and the Booth Multiplier at 15.75 ns. This optimized performance can be attributed to the specific sample operation used and the configuration of algorithms and data processed within the full adders and half adders.

Figure 18 presents a comparison of the debugging results for the three hardware accelerator multipliers, focusing on the total logical elements used, the registers utilized in combinational logic, and the input and output pins.

Specifications	Wallace	Booth parallel.	Baugh-Wolley
Flow Summary			
Flow Status	Successful - Mon Jul 17 18:12:38 2023	Successful - Mon Jul 17 18:20:43 2023	Successful - Mon Jul 17 18:16:29 2023
Quartus II 64-Bit Version	13.1.0 Build 162 10/23/2013 SJ Web Edition	13.1.0 Build 162 10/23/2013 SJ Web Edition	13.1.0 Build 162 10/23/2013 SJ Web Edition
Revision Name	wallace8	boothMult	BaughWooley8
Top-level Entity Name	wallace8	boothMult	BaughWooley8
Family	Cyclone IV E	Cyclone IV E	Cyclone IV E
Device	EP4CE22E22C6	EP4CE22E2217	EP4CE22F17C6
Timing Models	Final	Final	Final
Total logic elements	168 / 22,320 (< 1 %)	151 / 22,320 (< 1 %)	159 / 22,320 (< 1 %)
Total combinational functions	168 / 22,320 (< 1 %)	151 / 22,320 (< 1 %)	159 / 22,320 (< 1 %)
Dedicated logic registers	0 / 22,320 (0 %)	0 / 22,320 (0 %)	0 / 22,320 (0 %)
Total registers	0	0	0
Total pins	32 / 80 (40 %)	32 / 80 (40 %)	32 / 154 (21 %)
Total virtual pins	0	0	0
Total memory bits	0 / 608,256 (0 %)	0 / 608,256 (0 %)	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)	0 / 132 (0 %)	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)	0 / 4 (0 %)	0 / 4 (0 %)

Figure 18. Comparison of the summary flow of the multipliers.
Source: Own work.

After scrutinizing the comparison results regarding the logical elements utilized in the multipliers, several key conclusions emerge. Specifically, it was observed that the **Booth algorithm** (151) demonstrated more efficient utilization of the available logical elements when compared to the **Wallace algorithm** (168) and the **Baugh-Wooley algorithm** (159).

Description of Scrutinizing:

1. **Wallace Tree Multiplier (WTM)**
   - **Architecture**: The Wallace tree structure was used to reduce the partial product matrix. It was constructed as a network of adders that reduced the partial products in stages until a final sum was reached.
   - **Mode of Operation**: The partial products were generated, grouped, and reduced in multiple stages using half and full adders until only two rows remained. These rows were then summed to obtain the final product.
2. **Parallel Booth Multiplier (PBM)**
   - **Architecture**: The Booth encoding algorithm was employed to reduce the number of partial products. This involved examining adjacent bits and determining whether to add, subtract, or shift.
   - **Mode of Operation**: Booth encoding reduced the partial products, which were aligned and summed in parallel. A tree or array structure was used to efficiently combine these partial products.
3. **Baugh-Wooley Multiplier (BWM)**
   - **Architecture**: This multiplier was specifically designed to handle two's complement numbers. The architecture adjusted the partial products to ensure the final result was correct without additional steps for sign handling.
   - **Mode of Operation**: The partial products were adjusted during generation to ensure the sign bits were handled correctly. Then, the adjusted partial products were summed in a manner similar to other multipliers.

Table 1 shows a comparison of the algorithms based on their architecture, focusing on the reduction of partial products, handling of negative numbers, addition stages, and the use of half adders and full adders.

Table 1. Description of scrutinizing architectures.

	WTM (Wallace Tree Multiplier)	PBM (Parallel Booth Multiplier)	BWM (Baugh-Wooley Multiplier) Features
Partial Product Reduction	Yes	Yes (via Booth encoding)	Yes
Handling of Negative Numbers	Yes	Yes (via Booth encoding)	Yes (two's complement)
Reduction Structure	Wallace Tree	Tree or Array	Array or Linear Reduction
Stages of Addition	Multiple	Multiple	Multiple
Use of Adders	Half and Full	Half and Full	Half and Full

Source: Own work.

e-ISSN 2357-6014 / Vol. 20, no. 2 / may-august 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

In addition to exploring various multiplication algorithms, this research venture also examined high-level programming languages to assess their efficacy in implementing these algorithms. Beyond traditional hardware description languages like VHDL, Verilog, and SystemVerilog, the study expanded its scope to include popular languages known for their versatility, ease of use, and computational power. Python, MATLAB, Golang, C++, and Java were among the languages analyzed to evaluate their performance and suitability for executing multiplication algorithms. By broadening the analysis to include these diverse programming paradigms, the research aims to provide a comprehensive understanding of the interplay between algorithmic design and programming language selection in optimizing computational tasks.

Python, known for its simplicity, readability, and rich library ecosystem, as shown in Figure 19, exhibited a runtime of 100,130 ns. Despite being an interpreted language and generally slower than compiled languages, Python's ease of use and versatility make it a popular choice for prototyping and experimentation, particularly in academic and research environments.

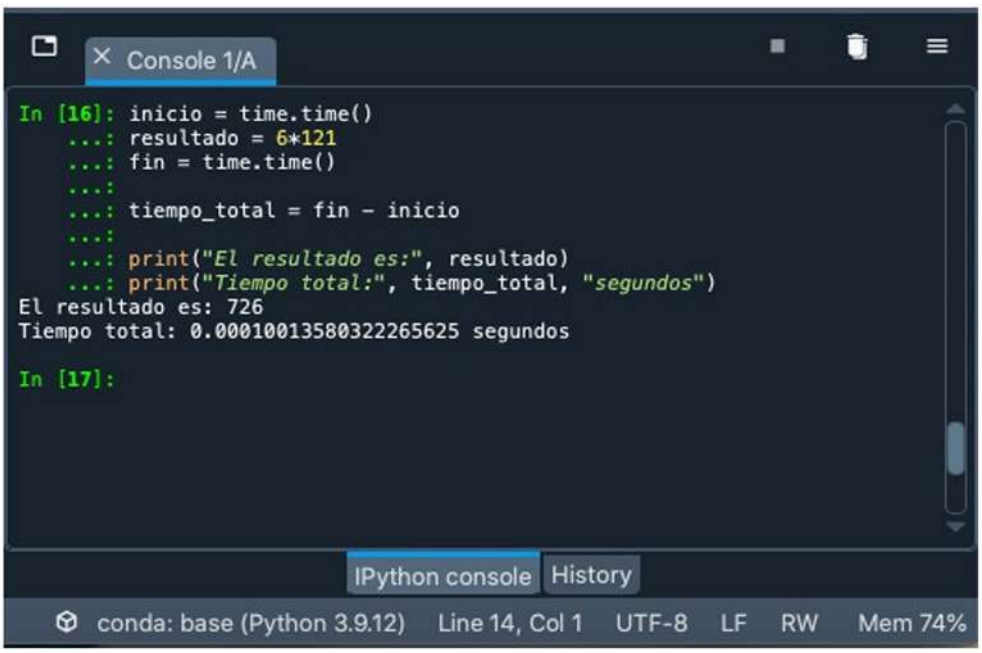

Figure 19. python excecution runtime code.
Source: Own work.

Java, valued for its platform independence, robustness, and extensive ecosystem, as shown in Figure 20, exhibited a runtime of 1,278 ns. While Java's execution speed may lag behind compiled languages like C++, its portability and vast library

support make it a preferred choice for building cross-platform applications and large-scale systems.

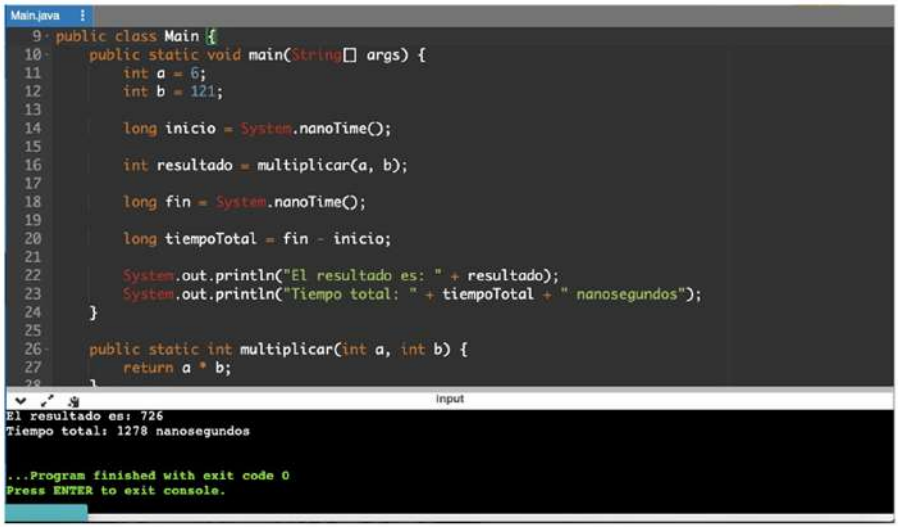

Figure 20. Java execution runtime code.
Source: Own work.

C++, renowned for its efficiency, low-level control, and performance, as shown in Figure 21, demonstrated a runtime of 160 ns. Leveraging features such as pointers, memory management, and optimized compilers, C++ remains a dominant choice for high-performance computing tasks where speed is paramount.

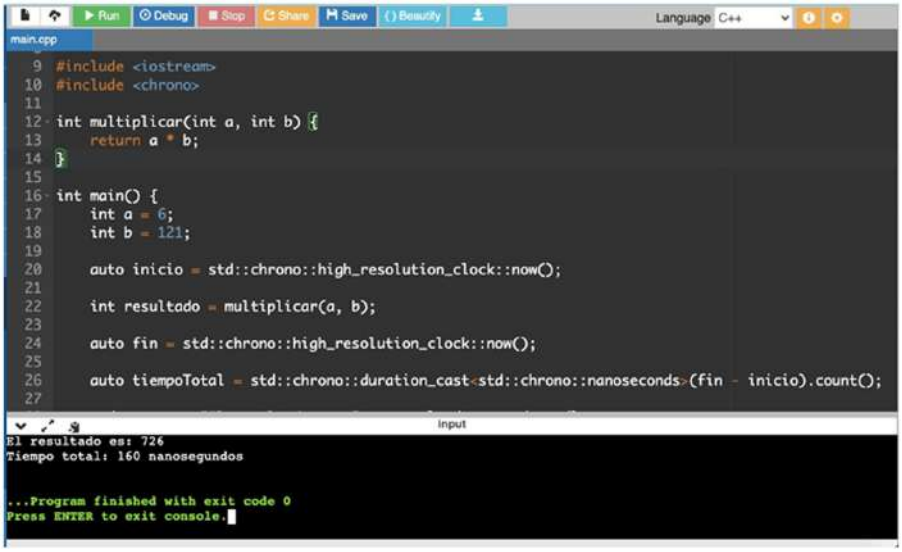

Figure 21. C++ execution runtime code.
Source: Own work.

MATLAB, renowned for its computational capabilities and extensive mathematical functions, as shown in Figure 22, demonstrated a runtime of 6,800 ns. This outcome highlights MATLAB's proficiency in handling numerical computations, making it a preferred choice for algorithmic research and development tasks where mathematical precision is paramount.

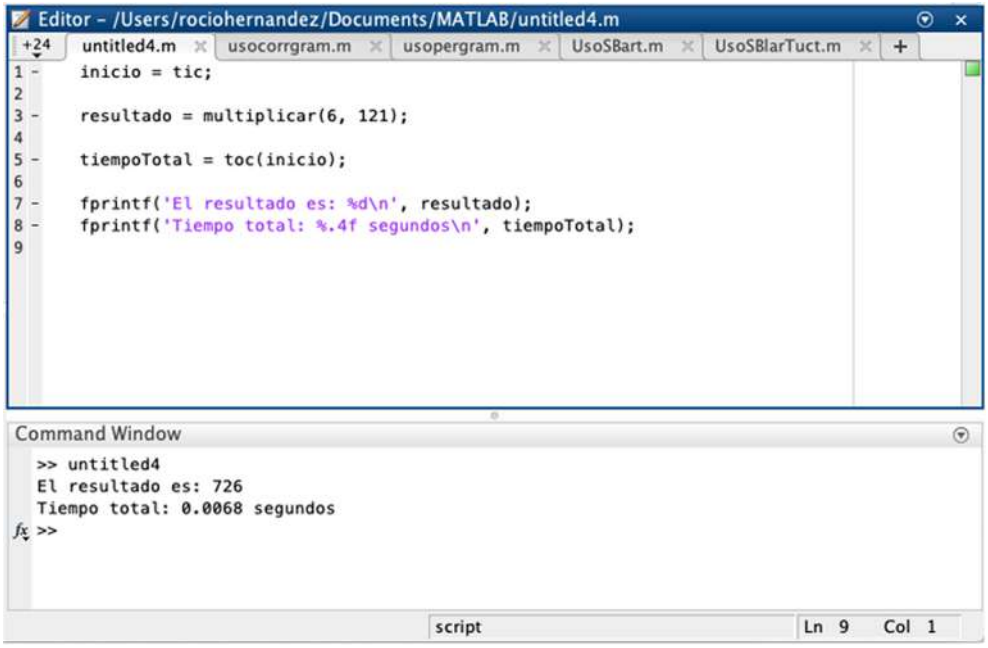

Figure 22. Matlab execution runtime code
Source: Own work.

Golang, recognized for its concurrency support and efficient runtime, as shown in Figure 23, recorded a swift runtime of 146 ns. Golang's compiled nature and lightweight concurrency mechanisms contribute to its suitability for performance-critical applications, including those involving complex computational tasks such as multiplication algorithms.

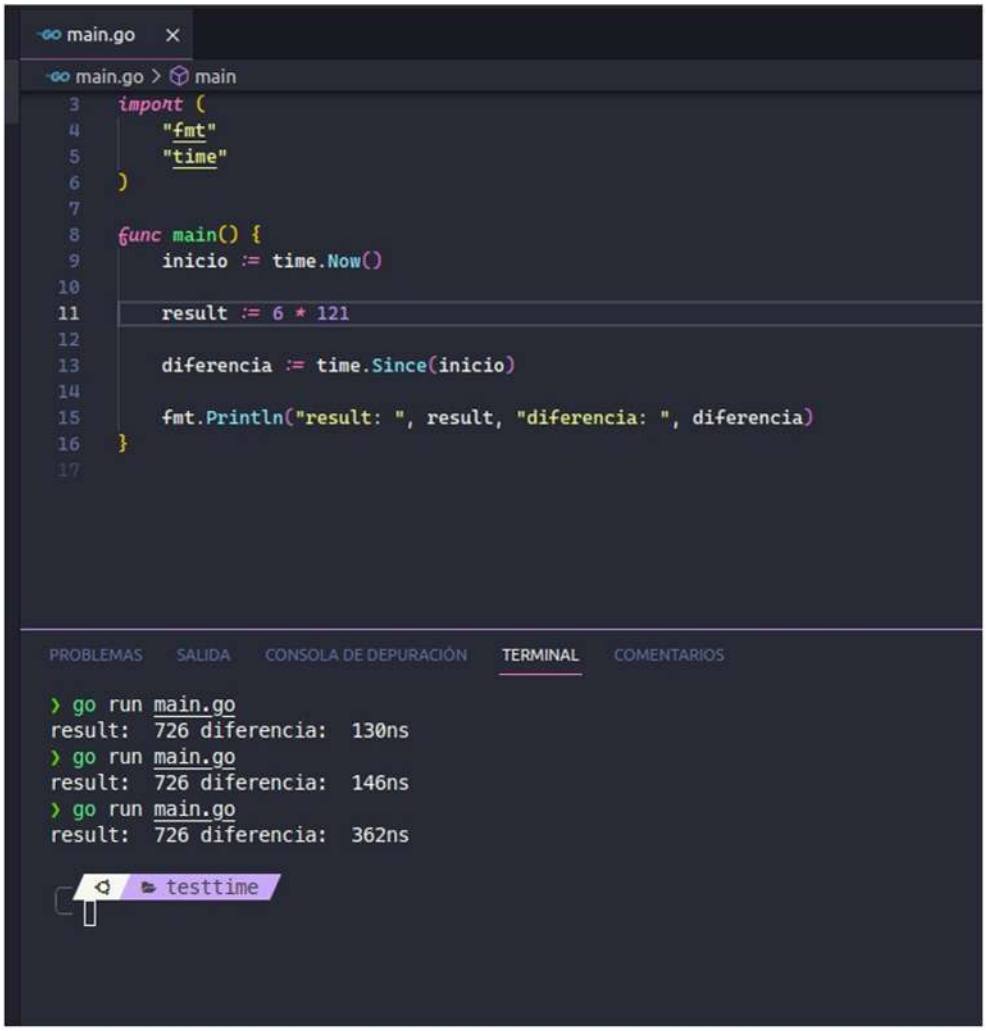

Figure 23. Golang execution runtime code.
Source: Own work.

The findings of this study are systematically organized and presented in a tabular format to facilitate clear comprehension and comparison. Through meticulous categorization, the results encapsulate crucial metrics such as execution times, algorithmic efficiency, and programming language performance. By employing this structured approach, readers can discern patterns, draw correlations, and gain insights into the relative strengths and weaknesses of each multiplication algorithm and programming language. Such an organized presentation not only enhances the accessibility of the research findings but also provides a framework for informed decision-making in future algorithmic implementations and language selections, as shown in Table 2.

Table 2. Comparison of multiplier performance against programming languages.

SOURCE	8 BIT OPERATION TIME
WALLACE TREE	13.03 nS
BOOTH PARALLEL	15.75 nS
BAUGH-WOLLEY	10.82 nS
PYTHON	100130 nS
C++	160 nS
MATLAB	6800 nS
GOLANG	146 ns
JAVA	1278 nS

Source: Own work.

The results from Table 2, analyzed in Figure 24, were derived from supporting code found in the GitHub repository within the file named 'multilanguage experiences.' When evaluating the multipliers in relation to operations performed in high-level programming languages, interesting insights emerge regarding their efficiency and ease of implementation. First and foremost, it is evident that hardware-implemented multipliers, such as Wallace, Booth, and Baugh-Wooley, outperform implementations in high-level programming languages in terms of speed. This can be attributed to the optimized nature of hardware multipliers and their ability to perform calculations in parallel.

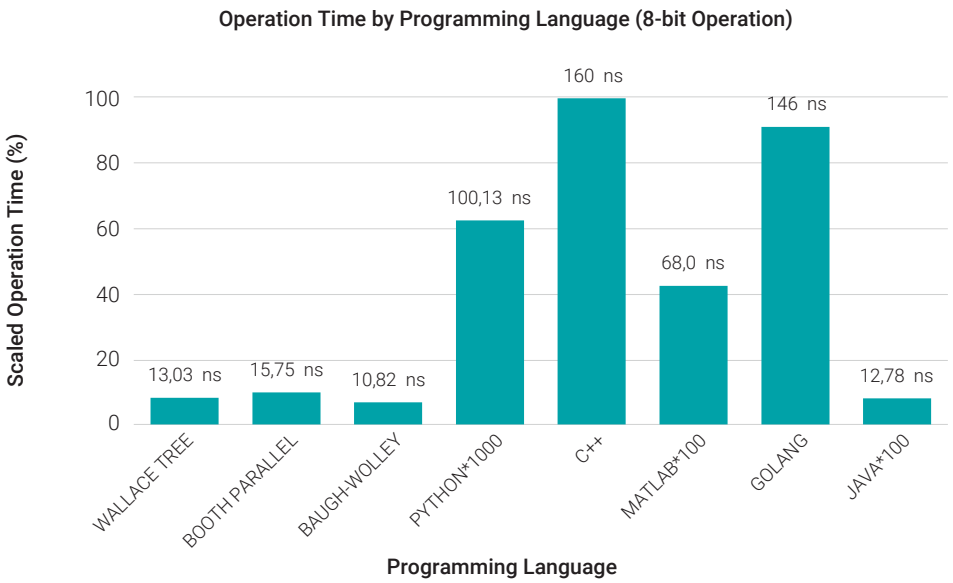

Figure 24. comparative bar graphic multipliers.
Source: Own work.

e-ISSN 2357-6014 / Vol. 20, no. 2 / may-august 2024 / Bogotá D.C., Colombia
Universidad Cooperativa de Colombia

The research involved generating a bar chart illustrating the scaled operation time by programming language for 8-bit operations. Each bar represented a specific programming language, including Wallace Tree, Booth Parallel, Baugh-Wooley, Python, C++, MATLAB, Golang, and Java. The height of each bar corresponded to the scaled operation time, providing a visual comparison of the efficiency of each programming language in executing the specified operations. This chart proved instrumental in offering insights into the relative performance of different programming languages, thereby supporting decision-making processes related to algorithm selection and implementation strategies within the field of digital hardware design.

4. DISCUSSION AND CONCLUSIONS

The project successfully enhanced the efficiency of the multiplication algorithms by utilizing various register shifts and multipliers tailored to specific operation cases. In terms of operation time, the Parallel Booth Multiplier (PBM) was the fastest, with an average time of 10.82 nanoseconds, followed by the Baugh-Wooley Multiplier (BWM) at 13.03 nanoseconds, and the Wallace Tree Multiplier (WTM) at 15.75 nanoseconds. Regarding the use of logical elements, the PBM again proved to be the most efficient, requiring only 151 elements, followed by BWM with 160 elements, and WTM with 168 elements.

Upon conducting a comparative analysis of multiplication algorithms for 8-bit systems—specifically the Wallace Tree Multiplier (WTM), Parallel Booth Multiplier (PBM), and Baugh-Wooley Multiplier (BWM)—several key insights emerged. Notably, although the PBM exhibited a slower operational speed compared to the other two algorithms, it demonstrated superior efficiency in terms of logical element utilization, using the fewest elements (151). Conversely, while the WTM had a mid-range operational speed, it required the highest number of logical elements (168). The BWM, however, emerged as the fastest of the three, striking a remarkable balance between operational speed and logical element efficiency, using only 159 elements. These findings underscore the multifaceted nature of algorithmic performance, highlighting the importance of considering both operational speed and resource utilization when evaluating and selecting multiplication algorithms for specific applications.

To further enhance multiplier speed, reducing the number of partial products in the modified Booth multiplier is crucial. Since multiplication involves a series of additions for these partial products, minimizing the number of such additions directly contributes to speed improvement.

The choice between serial or parallel operations during multiplication depends on the application. In parallel operations, accumulation is key, especially in domains like digital signal processing (DSP) and multimedia information processing. In these contexts, parallel algorithms are efficient because they perform simultaneous operations using more hardware, as opposed to the slower serial processing.

When comparing Booth algorithm-based multipliers, it's clear that the parallel implementation significantly outperforms the sequential version in terms of speed. However, the parallel multiplier performs well when both operands are positive or when one is positive and the other is negative, but it fails when both operands are negative. On the other hand, the sequential multiplier can handle both positive and negative numbers throughout the entire range, except for -128, where a sign error occurs.

From the results discussed, it can be concluded that the Baugh-Wooley Multiplier offers the best overall performance. It can handle both positive and negative numbers without restrictions and can be easily generalized for multiplications involving more bits.

5. DATA AVAILABILITY STATEMENT

Data associated with this paper and the described implementation are available under a creative common license in the GitHub of the Robotics and Industrial Automation Research, at the link https://github.com/AndresHernandezOrtega1/MULTIPLIERS.

6. REFERENCES

[1] S. Karunamurthi and K. Vijeyakumar, "A Novel n-Decimal Reversible Radix Binary-Coded Decimal Multiplier Using Radix Encoding Scheme," *Circuits, Systems, and Signal Processing*, vol. 40, pp. 3-6, 2021, doi: 10.1007/s00034-020-01549-w.

[2] S. Núñez Mejía, "Hidden Markov Models for early detection of cardiovascular diseases," *Ing. Solidar.*, vol. 20, no. 1, pp. 1–31, Dec. 2023, doi: 10.16925/2357-6014.2024.01.02.

[3] E. J. Rao, T. Ramanjaneyulu, and K. J. Kumar, "Advanced multiplier design and implementation using Hancarlson adder," pp. 1-6, 2018, doi: 10.1109/ICONIC.2018.8601252.

[4] Y. Chang et al., "A Low Power Radix-4 Booth Multiplier with Pre-Encoded Mechanism," *IEEE Access*, pp. 1-2, 2020, doi: 10.1109/ACCESS.2020.3003684.

[5] K. Saritha Raj et al., "Baugh-Wooley Multiplier design using Multiple Control Toffoli and Multiple Control Fredkin reversible logic gates," *International Review of Applied Sciences and Engineering*, vol. 14, no. 2, pp. 285-292, 2023, doi: 10.1556/1848.2022.00550.

[6] D. Gokulakrishan, R. Ramakrishnan, G. Saritha, and B. Sreedevi, "An advancing method for web service reliability and scalability using ResNet convolution neural network optimized with Zebra Optimization Algorithm," *Trans. Emerging Tel. Tech.*, vol. 35, no. 5, pp. 2-8, 2024, doi: 10.1002/ett.4968.

[7] N. Behera, M. Pradhan, and P. Mishro, "Analysis of Combinational Delay in Signed Binary Multiplier," in *2022 International Conference on Connected Systems & Intelligence (CSI)*, 2022, pp. 1-4, doi: 10.1109/CSI54720.2022.9924000.

[8] T. A. Rather et al., "Modelling and simulation of a reversible quantum logic based 4x3 multiplier design for nanotechnology applications," *Int. J. Theor. Phys.*, vol. 59, no. 1, pp. 57–67, 2020, doi: 10.1007/s10773-019-04285-3.

[9] V. Pala, V. Makhe, K. Bhuva, and R. Parekh, "RTL to GDSII Flow Implementation of 8-bit Baugh-Wooley Multiplier," in *2022 IEEE International Conference on Nanoelectronics, Nanophotonics, Nanomaterials, Nanobioscience & Nanotechnology (5NANO)*, 2022, pp. 1-6, doi: 10.1109/5nano53044.2022.9828876.

[10] C. K., D., V., and M. I., "Design of Power Delay Efficient Wallace Multiplier," in *2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2023, pp. 972-976, doi: 10.1109/ICACCS57279.2023.10112916.

[11] A. Jain, S. Bansal, S. Khan, S. Akhter, and S. Chaturvedi, "Implementation of an Efficient N×N Multiplier Based on Vedic Mathematics and Booth-Wallace Tree Multiplier," in *2019 International Conference on Power Electronics, Control and Automation (ICPECA)*, 2019, pp. 1-5, doi: 10.1109/ICPECA47973.2019.8975673.

[12] M. García-Parra, N. Plazas-Leguizamón, R. Colmenares-Cruz, N. Moreno-López, and A. Barrera-Siabato, "Technological surveillance of energy efficiency in agricultural production systems: a systematic review," *Portal SOAR: Sapienza Open Access Repository*, vol. 7, pp. 67-78, 2024, doi: 10.56183/soar.v7iEBOA7.39.

[13] Thamizharasan Viswanathan and P. Pavithra, "Design and analysis of Wallace tree multiplier using approximate full adder and kogge stone adder," *Ijireeice*, 2023, doi: 10.17148/IJIREEICE.2023.11303.

[14] A. Mokhtar, N. Zahari, C. Ping, M. Mustapha, N. Ismail, and S. Ismail, "Implementation of Modified Booth-Wallace Tree Multiplier in FPGA," *Journal of Computer Science & Computational Mathematics*, vol. 11, pp. 49-52, 2021, doi: 10.20967/jcscm.2021.03.003.

[15] B. Mukherjee and A. Ghosal, "Design and Analysis of a Low Power High-Performance GDI based Radix 4 Multiplier Using Modified Booth Wallace Algorithm," in *2018 IEEE Electron Devices Kolkata Conference (EDKCON)*, 2018, pp. 247-251, doi: 10.1109/EDKCON.2018.8770494.

[16] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "RoBA Multiplier: A Rounding-Based Approximate Multiplier for High-Speed yet Energy-Efficient Digital Signal Processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, pp. 393-401, 2017, doi: 10.1109/TVLSI.2016.2587696.

[17] Y. Luo, Y. Wang, H. Sun, Y. Zha, Z. Wang, and H. Pan, "CORDIC-Based Architecture for Computing Nth Root and Its Implementation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, pp. 4183-4195, 2018, doi: 10.1109/TCSI.2018.2835822.

[18] A. Mokhtar, S. A. Karim, S. P. Chew, S. M. F. S. M. Dardin, L. S. Supian, and F. Hashim, "FPGA implementation of CORDIC algorithm in digital modulation," *Journal of Fundamental and Applied Sciences*, vol. 9, pp. 279, 2018, doi: 10.4314/jfas.v9i3s.23.

[19] M. Heidarpur et al., "CORDIC-SNN: On-FPGA STDP Learning With Izhikevich Neurons," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 7, pp. 2235-2245, 2019, doi: 10.1109/TCSI.2019.2899356.

[20] A. Boettcher and M. Kumm, "Towards Globally Optimal Design of Multipliers for FPGAs," *IEEE Transactions on Computers*, pp. 1–13, Jan. 2023, doi: 10.1109/TC.2023.3238128.

[21] D. Perišić, "NEW KIND OF IIR DIGITAL FILTERS INTENDED FOR PULSE PERIOD FILTERING," *Revue Roumaine des Sciences Techniques, Série Électrotechnique et Énergétique*, vol. 69, pp. 61-66, 2024, doi: 10.59277/RRST-EE.2024.1.11.

[22] S. Raveendran, P. Edavoor, N. Balachandra, and M. Vasantha, "Inexact Signed Wallace Tree Multiplier Design Using Reversible Logic," *IEEE Access*, vol. 9, pp. 108119-108130, 2021, doi: 10.1109/ACCESS.2021.3100892.

[23] M. H. Haider and S.-B. Ko, "Booth Encoding-Based Energy Efficient Multipliers for Deep Learning Systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 6, pp. 2241-2245, June 2023, doi: 10.1109/TCSII.2022.3233923.

[24] B. V. Dharani et al., "Booth Multiplier: The Systematic Study," in *ICCCE 2020*, Springer, Singapore, 2021, pp. 28-33, doi: 10.1007/978-981-15-7961-5_88.

[25] B. Mahesh and T. Srivasarao, "Performance Evaluation of FFT through Adaptive Hold Logic (AHL) Booth Multiplier," in *2023 International Conference for Advancement in Technology (ICONAT)*, 2023, pp. 1-6, doi: 10.1109/ICONAT57137.2023.10080290.

[26] P. Singh and M. Kumar, "Design of Partial Product Generator Circuit for Approximate Radix-8 Booth Multiplier with Lower Delay," in *VLSI, Microwave and Wireless Technologies*, Springer, Singapore, 2023, pp. 1-1, doi: 10.1007/978-981-19-0312-0_53.

[27] K. Rayudu, D. Jahagirdar, and P. Rao, "Design and testing of systolic array multiplier using fault injecting schemes," *Computer Science and Information Technologies*, vol. 3, no. 1, pp. 1-9, 2022, doi: 10.11591/csit.v3i1.p1-9.

[28] V. Pala, V. Makhe, K. Bhuva, and R. Parekh, "RTL to GDSII Flow Implementation of 8-bit Baugh-Wooley Multiplier," in *2022 IEEE International Conference on Nanoelectronics, Nanophotonics, Nanomaterials, Nanobioscience & Nanotechnology (5NANO)*, 2022, pp. 1-6, doi: 10.1109/5nano53044.2022.9828876.

[29] V. Sm and K. Suresh, "An Efficient Design Approach of ROI Based DWT Using Vedic and Wallace Tree Multiplier on FPGA Platform," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 4, pp. 2433-2442, 2019, doi: 10.11591/ijece.v9i4.pp2433-2442.

[30] R. Gayathri et al., "VLSI Design of Approximate Baugh-Wooley Multiplier for Image Edge Computing," *Naksh Solutions 28-33 International Journal of Advanced Research in Science, Communication and Technology*, ISSN: 2581-9429, 2023, doi: 10.48175/IJARSCT-9537.

[31] A. Sadeghi, N. Shiri, M. Rafiee, and M. Tahghigh, "An efficient counter-based Wallace-tree multiplier with a hybrid full adder core for image blending," *Frontiers of Information Technology & Electronic Engineering*, vol. 23, pp. 950 - 965, 2022, doi: 10.1631/FITEE.2100432.

[32] F. A. Escobar Revelo, V. H. Mosquera Leyton, and C. F. Rengifo Rodas, "Tomografía de impedancia eléctrica: fundamentos de hardware y aplicaciones médicas," *Ing. Solidar.*, vol. 16, no. 3, Sep. 2020, doi: 10.16925/2357-6014.2020.03.02.

[33] M. Saha and A. Dandapat, "Modified Baugh Wooley Multiplier using Low Power Compressors," in *2021 2nd International Conference for Emerging Technology (INCET)*, 2021, pp. 1-6, doi: 10.1109/INCET51464.2021.9456141.

[34] B. Zhang, Z. Cheng, and M. Pedram, "A High-Performance Low-Power Barrett Modular Multiplier for Cryptosystems," in *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2021, pp. 1-6, doi: 10.1109/ISLPED52811.2021.9502490.

[35] B. Jeevan, P. Samskruthi, P. Sahithi, and K. Sivani, "Implementation of parallel multiplier based on Booth computing method using FPGA," in *2022 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI)*, 2022, pp. 1-8, doi: 10.1109/ACCAI53970.2022.9752479.

[36] P. Karuppusamy, "Design and Analysis of Low-Power, High-Speed Baugh Wooley Multiplier," *Journal of Electronic Imaging*, 2019, pp. 60-70, doi: 10.36548/jei.