

2220588- Electronic II

ACDA6001

by Luis Lloyd

Submission date: 24-Apr-2025 09:54PM (UTC+0100)

Submission ID: 256536942

File name: 42507_Luis_Lloyd_2220588-_Electronic_II_ACDA6001_654553_1673589917.docx (6.07M)

Word count: 5780

Character count: 31521



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Electronics II

Module: ACDA6001

**Critical Evaluation of Braun's Multiplier and Wallace Tree
Multiplier Using VHDL**

Student Number: 2220588

Name: Luis Lloyd

Contents

Contents.....	2
Figures	3
List of Tables.....	5
Abbreviations	6
Abstract:	6
Introduction:	6
Background Research:.....	6
Braun Multiplier	7
Wallace Tree Multiplier	7
Hardware Description Language and FPGA Implementation	7
Comparison Factors.....	7
Applications and Advantages of Multipliers:	8
System Design Flow Chart/ pseudocode:	8
Brauns Pseudocode	9
Wallace Tree pseudo code.....	9
Methodologies:.....	10
Understanding VHDL and FPGA Design	10
Design and Implementation Stages	11
1. Initial Ladder Diagrams and Planning	11
2. Component Design in VHDL	11
3. Multiplier Implementation in VHDL	12
4. Simulation and Testing	13
5. FPGA Implementation and Performance Reports	13
Multipliers Block Diagram and Algorithms	15
Braun Multiplier Block Diagram.....	15
Wallace Tree Multiplier Block Diagram	17
Algorithms.....	17
VHDL Code:.....	18
Timing diagram & Schematic design:.....	18

Braun Multiplier – Timing and Schematics	19
Wallace Tree Multiplier – Timing and Schematics.....	21
FPGA Design Screen Dumps with Pin Allocation:	23
Simulation Results:.....	37
Comparison, analysis & Interpretation of results:	42
Time Analysis	42
Test Accuracy and Functionality.....	43
Design Efficiency and Use Case Comparison	43
Interpretation of Power Utilization and LUTs:	44
Braun Multiplier – Power Breakdown.....	44
Braun Multiplier – LUT Usage	44
Wallace Tree Multiplier – Power Breakdown	45
Wallace Tree – LUT Usage	46
Details of testing and future improvements:	47
Future considerations	49
Conclusion:	50
References:.....	50
Appendix	52
Brauns Code	52
Wallace Tree Code	57
Hardware LED Photos	60

Figures

Figure 1. Braun Multiplier Operation Flowchart.....	9
Figure 2. Wallace Tree Multiplier Operation Flowchart.....	10
Figure 3 VHDL Code of Full Adder Structure	11
Figure 4 VHDL Code of Half Adder structure	11
Figure 5 Braun 8x8 Multiplier Algorithm	12
Figure 6 Wallace Tree 8x8 Algorithm	13
Figure 7 Braun Multiplier Block Diagram	16
Figure 8 Wallace Tree Multiplier Block Diagram	17
Figure 9 Constraint file defining a virtual clock and timing constraints used for critical path analysis.	18
Figure 10 Braun Schematic Vivado.....	19
Figure 11 Braun longest path	20
Figure 12 Timing constrains worst negative slack Braun	20
Figure 13 Wallace Tree Schematic	21

Figure 14 Wallace Tree Longest Path	22
Figure 15 Timing constrains worst negative slack Wallace	23
Figure 16 Vivado floorplan for the Braun Multiplier showing block placement across FPGA tiles.	23
Figure 17 FPGA routing layout showing signal interconnections for the Braun Multiplier.	24
Figure 18 Brauns Resource utilization summary.	24
Figure 19 Pin allocation mapping inputs to switches and outputs to LEDs.....	25
Figure 20 Braun Multiplier deployed on Basys 3 board with live output shown via LEDs (FFxFF).	25
Figure 21 Vivado floorplan for the Wallace Tree	26
Figure 22 FPGA routing layout showing signal interconnections for the Braun Multiplier.	26
Figure 23 Wallace Tree Resource Utilization Summary.	27
Figure 24 Wallace Tree Multiplier deployed on Basys 3 board with live output shown via LEDs (08x05).....	27
Figure 25 RTL schematic of Braun multiplier post-synthesis.....	28
Figure 26 RTL schematic of Wallace Tree multiplier post-synthesis	28
Figure 27 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	29
Figure 28 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	30
Figure 29 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	30
Figure 30 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	31
Figure 31 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	31
Figure 32 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	32
Figure 33 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	32
Figure 34 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	32
Figure 35 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	33
Figure 36 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	33
Figure 37 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	34
Figure 38 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	35

Figure 39 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	35
Figure 40 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	36
Figure 41 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)	36
Figure 42 Braun8x8 Multiplier Lowest Value 08 x 05 = 0028	37
Figure 43 Figure 43 Braun8x8 Multiplier Mid value 1 Braun AB x BA = 7c3e	38
Figure 44 Braun8x8 Multiplier Mid value 2 AC x DC= 93d0	38
Figure 45 Braun8x8 Multiplier High Range value Braun FF x FF = fe01	39
Figure 46 Wallace tree Multiplier Lowest Value 08 x 05 = 0028.....	39
Figure 47 Wallace tree Multiplier Mid Valve 1 AB x BA = 7c3e	40
Figure 48 Wallace tree Multiplier Mid Valve 2 AC x DC = 93d0	40
Figure 49 Wallace tree Multiplier High Value ff x ff = fe01	41
Figure 50 Power Report for Braun Multiplier	44
Figure 51 Braun Utilization LUT	45
Figure 52 Power Report Wallace Tree Multiplier	46
Figure 53 Wallace Tree Resource utilization summary.	46
Figure 54 side by side comparison.....	47
Figure 55 Hardware test outputs from the Basys 3 board for multiple 8x8 input combinations	48
Figure 56 Braun Code.....	52
Figure 57 Braun code.....	53
Figure 58 Braun code.....	54
Figure 59 Braun code.....	55
Figure 60 Braun code.....	56
Figure 61 Wallace Tree Code	57
Figure 62 Wallace Tree Code	58
Figure 63 Wallace Tree Code	59
Figure 64 Wallace Tree code	60
Figure 65 AB x BA Hardware Image	61
Figure 66 AC x DC Hardware Image.....	61

List of Tables

Table 1 Comparison table of multipliers	8
Table 2: Expected Out Values.....	15
Table 3 Results of Simulation testing for both multipliers	41
Table 4: Timing Evaluation.....	40
Table 5 LUT Comparison	43
Table 6 Power Report Comparison	47
Table 7 Comparison of Braun and Wallace Tree Multipliers	49

Abbreviations

- **DSP** – Digital Signal Processing
- **VHDL** – VHSIC Hardware Description Language
- **FPGA** – Field-Programmable Gate Array
- **LUT** – Look-Up Table
- **IO** – Input/Output
- **HA** – Half Adder
- **FA** – Full Adder
- **WNS** – Worst Negative Slack
- **ALU** – Arithmetic Logic Unit
- **RTL** – Register Transfer Level

Abstract:

This report presents the VHDL-based design, simulation, and evaluation of Braun and Wallace Tree multipliers. Implemented on an FPGA, both designs are analyzed for performance across delay, power usage, and hardware efficiency. Wallace Tree consistently outperformed Braun in timing metrics, while Braun proved more resource efficient highlighting architectural trade-offs.

Introduction:

Multipliers are key components in digital systems, used in many applications, such as image processing, DSP, and cryptography. As multiplication is more complex than addition or subtraction, designing efficient multipliers is crucial to improving performance, reducing delays, and minimizing power consumption.

This report compares two architectures:

Braun's Multiplier: A simple, array-based design that adds partial products sequentially.

Wallace Tree Multiplier: A faster design that compresses partial products using a tree of full and half adders.

Both designs are implemented in VHDL, simulated and synthesized using Xilinx Vivado, and deployed on FPGA. Their performance is based on delay, power, and hardware utilization [1].

Background Research:\

Braun Multiplier

Introduced by B. Braun in 1973 [2], the Braun multiplier is an array-based architecture for unsigned binary multiplication. It forms partial products using AND gates and adds them row by row through a regular grid of adders. This structure is simple, highly regular, and easy to implement in hardware, particularly in VLSI design. However, because all partial products are summed sequentially, it results in longer critical path delays as operand sizes grow [3]. Despite this, the Braun multiplier is ideal for low-power, resource-constrained systems like embedded microcontrollers, basic digital clocks, and calculator ICs.

Wallace Tree Multiplier

Developed by C. S. Wallace in 1964 [4], the Wallace Tree multiplier reduces critical path delay by organizing partial products into layered stages of full and half adders. This tree structure compresses additions in parallel, achieving logarithmic depth relative to operand size [5]. It offers significantly faster performance than array-based designs and is widely used in systems requiring high throughput, such as GPUs, cryptographic accelerators, and telecom DSPs.

The Wallace Tree multiplier, developed by C.S. Wallace in 1964 [4], uses a layered approach of adders to compress partial products in parallel, significantly reducing critical path delays. It has become the preferred architecture in high-performance DSP and cryptographic processors. Modern FPGA comparisons [5] confirm Wallace Tree's superior speed, especially under pipelined configurations, albeit with increased logic utilization.

Hardware Description Language and FPGA Implementation

Both multiplier architectures are implemented in this study using VHDL. They are simulated and synthesized using Vivado, targeting a Basys 3 FPGA board. Vivado enables performance validation through simulation, resource analysis, and hardware deployment.

Comparison Factors

- This report compares both designs using the following criteria:
- **Delay** – Time to produce a final product
- **Power Consumption** – Efficiency during active use
- **Resource Usage** – FPGA utilization (LUTs, slices, IOs)

Table 1 below illustrates the key differences.

	Braun Multiplier	Wallace Tree Multiplier	Notes
Delay	Higher	Lower	Wallace Tree reduces addition stages.
Power Consumption	Lower	Slightly Higher	Braun is more power-efficient in basic designs.
Resource Usage	Lower	Higher	Wallace Tree uses more FPGA logic.
Application Type	Basic, low-speed systems	High-speed, performance-critical systems	Choice depends on required performance vs. resource limits.

Table 1 Comparison table of multipliers

Applications and Advantages of Multipliers:

Multipliers are widely used in various digital systems. They are essential in areas that deal with high-speed calculations and large volumes of data. The choice of multiplier architecture can directly affect the performance, efficiency, and reliability of the system.



Braun multipliers are often used in simpler systems where regular structure and ease of implementation are more important than speed. These are suitable for systems like basic embedded devices, digital signal generators, and some image filtering tasks where lower power and predictable delay are acceptable [6].

Wallace Tree multipliers are more suited for performance-critical applications. Because of their ability to reduce the number of sequential addition stages, they are ideal for use in high-speed digital signal processing (DSP), graphics processing units (GPUs), and cryptographic hardware like RSA encryption engines [7].

Both are essential in FPGA workflows, where selecting the right architecture impacts the system's efficiency and scalability [8].

System Design Flow Chart/ pseudocode:

Visual tools such as flowcharts and pseudocode provide a clear overview of the system design process by breaking down complex logic into manageable steps. These representations help during development and debugging, especially when mapping the logical structure to VHDL implementation.

Figure 1 shows the operational flow for the Braun multiplier, highlighting its sequential handling of partial products. Figure 2 illustrates the Wallace Tree approach, where partial products are reduced in parallel stages for faster processing.

Brauns Pseudocode

1. Start
2. Load A 8-bit input
3. Load B 8-bit input
4. Generate all partial products using AND gates
5. Align partial products according to bit significance
6. Use array of adders to sum the partial products
7. Output final 16-bit product
8. End.

A simplified flow chart of the Braun operation displayed in figure 1.

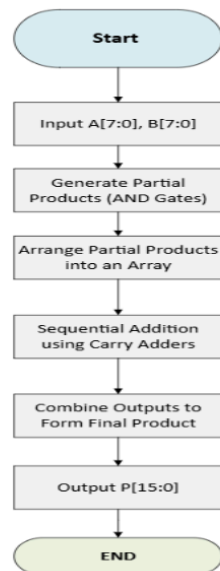


Figure 1. Braun Multiplier Operation Flowchart

Wallace Tree pseudo code

1. Start
2. Load A 8-bit input
3. Load B-8-bit input
4. Generate all partial products using AND gates
5. Group partial products
6. Use full and half adders to reduce each group

7. Repeat reduction until only 2 rows remain
8. Add final two rows using carry adder
9. Output final 16-bit product
10. End.

Wallace tree flow chart of the operation displayed in figure.2.

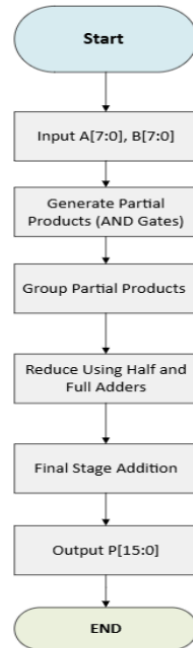


Figure 2. Wallace Tree Multiplier Operation Flowchart

Methodologies:

This project involved designing and evaluating two 8x8-bit unsigned binary multipliers using VHDL and Vivado Design Suite, focusing on delay, power consumption, and FPGA resource usage. Both designs accept two 8-bit inputs and produce a 16-bit output.

Understanding VHDL and FPGA Design

VHDL is used to describe, simulate, and synthesize digital logic. A structural VHDL approach was adopted in this project, using Full Adders (FAs) and Half Adders (HAs) to better reflect physical hardware behavior. This method avoids abstraction and provides full control over internal logic [9].

Design and Implementation Stages

The process was broken into five clear stages:

1. Initial Ladder Diagrams and Planning

Initial ladder-style diagrams were sketched to plan the logic layout for each multiplier. These helped map partial products and adder arrangements before coding.

2. Component Design in VHDL

Basic building blocks — Full Adders and Half Adders — were implemented as reusable VHDL components using port mapping. These can be shown in figures 3 and 4.



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Fulladder is
    Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          CIN : in STD_LOGIC;
          S : out STD_LOGIC;
          COUT : out STD_LOGIC);
end Fulladder;

architecture gate_level of Fulladder is
begin
    S end gate_level: <= A XOR B XOR CIN;
    COUT <= (A AND B) OR (CIN AND A) OR (CIN AND B);
end;
```

Figure 3 VHDL Code of Full Adder Structure

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Halfadder is
    Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          S : out STD_LOGIC;
          COUT : out STD_LOGIC);
end Halfadder;

architecture gate_level of Halfadder is
begin
    S <= A XOR B;
    COUT <= A AND B;
end gate_level;
```

Figure 4 VHDL Code of Half Adder structure

3. Multiplier Implementation in VHDL

Both multipliers were designed in structural VHDL:

- **Braun Multiplier** uses sequential rows of FA and HA to add aligned partial products.
- **Wallace Tree Multiplier** compresses the products through parallel reduction layers.

Both architectures were implemented using manual signal mapping, closely following the architecture diagrams.

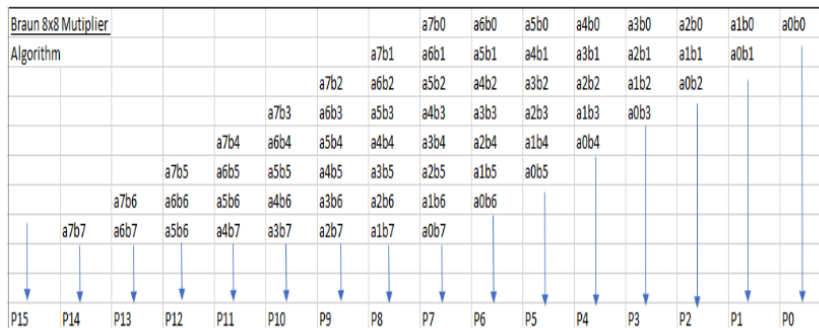


Figure 5 Braun 8x8 Multiplier Algorithm



7

Each design was verified using custom testbenches in Vivado. Identical 8-bit input values were used for both architectures to ensure consistency. The chosen test cases represented a range of binary values to evaluate multiplier behavior under low, medium, and high input conditions:

- 08 x 05 (Low)
- AB x BA (Mid-range 1)
- AC x DC (Mid-range 2)
- FF x FF (High)

Results were checked to confirm correct 16-bit output generation for all cases.

5. FPGA Implementation and Performance Reports

Post-simulation, both designs were synthesized for the Basys 3 board. Vivado reports were used to analyze:

- **Delay**
- **Power consumption**
- **Resource usage** (LUTs, flip-flops, slices)

This data allowed a direct comparison between the Braun and Wallace designs under the same conditions.

Multipliers Block Diagram and Algorithms

Block diagrams were created to illustrate the architecture and logic flow of both the Braun and Wallace Tree multipliers. They provide a clear overview of the data path and internal structure used in each design, supporting VHDL development and hardware implementation.

Braun Multiplier Block Diagram

The Braun multiplier uses an array-style structure consisting of AND gates, Half Adders (HA) in green, and Full Adders (FA) in blue. It adds the partial products row by row, aligning each according to its binary weight. This design makes it easier to understand and implement in hardware but comes with higher propagation delay due to the number of adder stages, as shown in Figure 7.

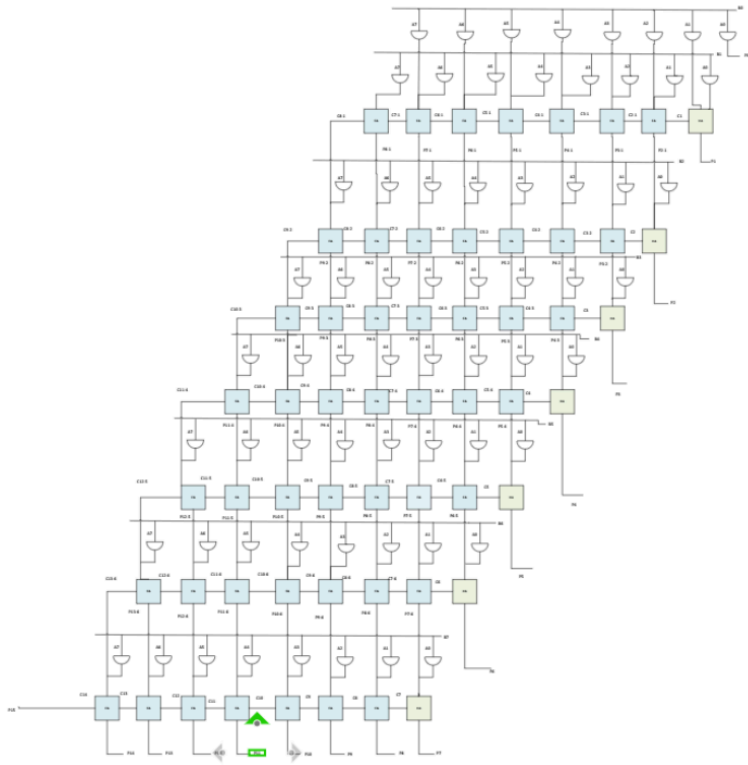


Figure 7 Braun Multiplier Block Diagram

Wallace Tree Multiplier Block Diagram

The Wallace Tree multiplier improves performance by reducing sequential additions. Like Braun, it generates partial products with AND gates but compresses them using layered Full and Half Adders. This reduces the critical path delay and boosts speed, making it suitable for high-performance use cases, as shown in Figure 8.

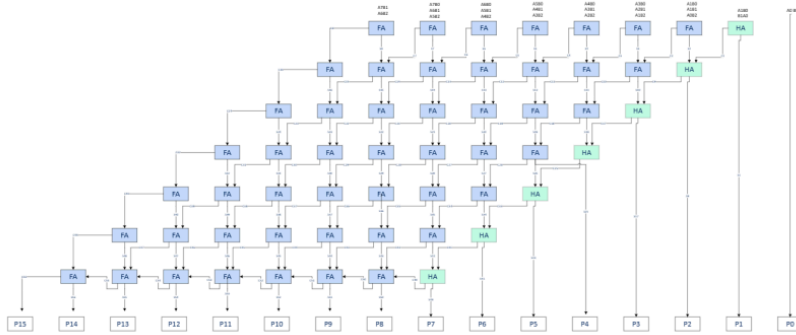


Figure 8 Wallace Tree Multiplier Block Diagram

Algorithms

After outlining the high-level architecture of both multiplier designs, this section demonstrates their logical functionality. The following truth tables show how each multiplier processes binary inputs to generate a correct 16-bit product.

The Braun multiplier uses a sequential accumulation of partial products, while the Wallace Tree uses a layered reduction approach to minimize delay. Table 2 below shows the expected output.

Input A (Hex)	Input B (Hex)	Expected Product (Hex)	Binary Output
08	05	0028	0000 0000 0010 1000
AB	BA	7D56	0111 1101 0101 0110
AC	DC	95A8	1001 0101 1010 1000
FF	FF	FE01	1111 1110 0000 0001

Table 2 Expected out Values

VHDL Code:

Both multiplier designs were implemented using a modular VHDL approach for clarity and reusability. The main components included:

- **Half Adder** – Produces a sum and carry from two input bits.
- **Full Adder** – Adds two bits plus a carry-in.
- **Multiplier Module** – Arranged using either (Braun) or (Wallace).

Full VHDL code is provided in the Appendix.



Timing diagram & Schematic design:

Timing analysis is essential in digital system design. It determines how long data takes to propagate through the circuit and sets the upper limit on the system's clock frequency. The critical path, the longest combinational logic path from input to output, limits overall performance.

To ensure accurate analysis, a **virtual clock** and I/O delay constraints were defined in Vivado. This helped the tool detect the actual **worst negative slack (WNS)** and validate design closure.

```
1 # Create a virtual clock (needed for timing analysis)
2
3 create_clock -period 45.000 -name virtual_clk
4
5 # Set delays based on analytical calculations
6 set_input_delay -clock virtual_clk -max 1.000 [get_ports A*]
7 set_input_delay -clock virtual_clk -max 1.000 [get_ports B*]
8 set_output_delay -clock virtual_clk -max 2.000 [get_ports P*]
9
10 # Set max delay for the critical path (longest delay)
11 set_max_delay -from [get_ports A*] -to [get_ports P*] 41.000
12
13 set_property IOSTANDARD LVCMOS33 [get_ports {A[7]}]
14 set_property IOSTANDARD LVCMOS33 [get_ports {A[6]}]
15 set_property IOSTANDARD LVCMOS33 [get_ports {A[5]}]
```

Figure 9 Constraint file defining a virtual clock and timing constraints used for critical path analysis.

Braun Multiplier – Timing and Schematics

The Braun architecture uses a regular, layered structure of AND gates followed by Full and Half Adders. This sequential approach increases the depth of logic and leads to a longer critical path.

Figure 10 shows the synthesized schematic of the Braun multiplier, while Figure 11 highlights the critical path in red.

Due to its regular and arrayed layout, the Braun architecture accumulates delay across several logic levels. This significantly increases the depth of the critical path.

Manual Delay Estimate:

The Braun multiplier includes 1 AND gate followed by an estimated 14 adder stages.

$$1 \text{ ns (AND gate)} + 14 \times 2 \text{ ns (FA/HA)} = 29 \text{ ns}$$

Vivado reports a worst negative slack (WNS) of approximately **13.5 ns**, shown in figure 12 This shows that the critical path is shorter than the estimated, and all timing constraints have been met, confirming that the design operates within its defined clock period [10].

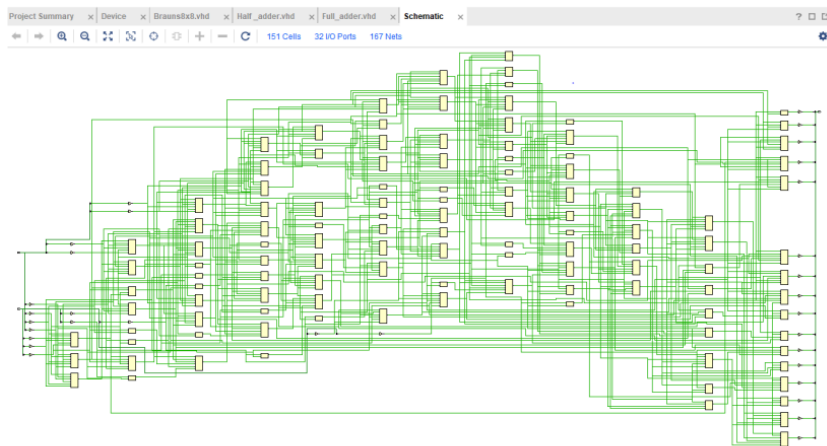


Figure 10 Braun Schematic Vivado

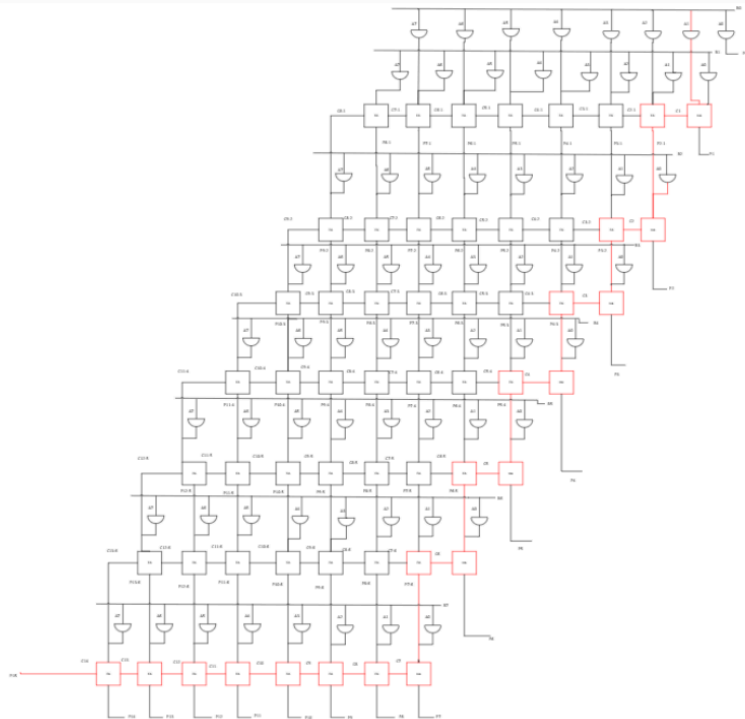


Figure 11 Braun longest path

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 13.508 ns	Worst Hold Slack (WHS): NA	Worst Pulse Width Slack (WPWS): NA
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): NA	Total Pulse Width Negative Slack (TPWS): NA
Number of Failing Endpoints: 0	Number of Failing Endpoints: NA	Number of Failing Endpoints: NA
Total Number of Endpoints: 16	Total Number of Endpoints: NA	Total Number of Endpoints: NA

All user specified timing constraints are met.

Figure 12 Timing constrains worst negative slack Braun

Wallace Tree Multiplier – Timing and Schematics

The Wallace Tree architecture improves speed by reducing the depth of its adder stages. Partial products are grouped and reduced in a tree-like structure, allowing for more parallel processing and fewer sequential delays.

The synthesized schematic is shown in Figure 13, and the critical path is highlighted in red in Figure 14, clearly shorter than Braun's.

Given that AND gate delay = 1ns

Full adder/half adder delay = 2ns

$$1 \text{ ns (AND gate)} + 20 \times 2 \text{ ns (FA/HA)} = 41 \text{ ns}$$

Manual delay estimation is trickier due to the irregular structure, but Vivado reports a WNS of 6.09 ns (Figure 15), confirming the expected performance advantage over the Braun design [11].

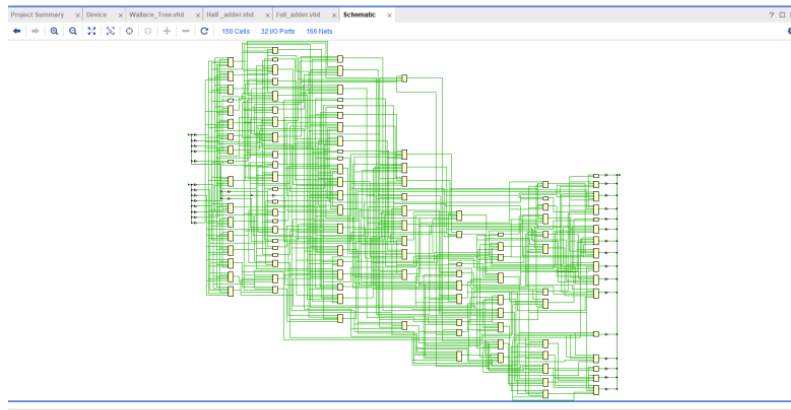


Figure 13 Wallace Tree Schematic

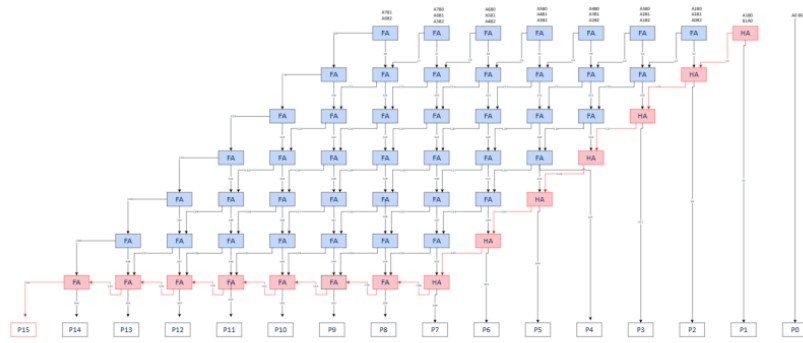


Figure 14 Wallace Tree Longest Path

13

Design Timing Summary			
Setup	Hold	Pulse Width	
Worst Negative Slack (WNS): 6.090 ns	Worst Hold Slack (WHS): NA	Worst Pulse Width Slack (WPWS): NA	
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): NA	Total Pulse Width Negative Slack (TPWS): NA	
Number of Failing Endpoints: 0	Number of Failing Endpoints: NA	Number of Failing Endpoints: NA	
Total Number of Endpoints: 16	Total Number of Endpoints: NA	Total Number of Endpoints: NA	
All user specified timing constraints are met.			

Figure 15 Timing constraints worst negative slack Wallace

FPGA Design Screen Dumps with Pin Allocation:

The Braun Multiplier was successfully synthesized and implemented on the Basys 3 development board using Vivado. Inputs A and B (8-bit each) were mapped to switches 0–15, while the 16-bit output was displayed using the onboard LEDs. All timing constraints were met, and the hardware worked as expected.

Figure 16 shows the Vivado floorplan of the Braun design, displaying how logic blocks were placed across the FPGA fabric.



Figure 16 Vivado floorplan for the Braun Multiplier showing block placement across FPGA tiles.

Figure 17 presents the routing layout, confirming all logic interconnects were correctly wired.

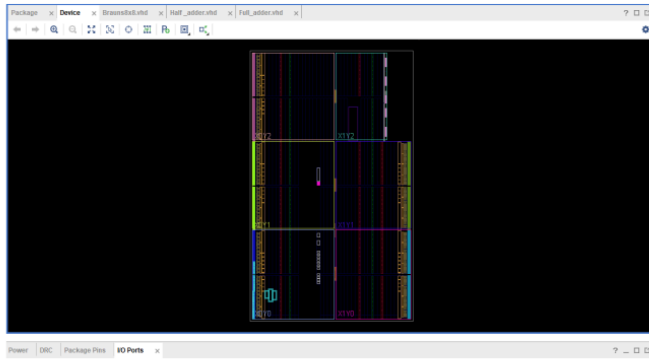


Figure 17 FPGA routing layout showing signal interconnections for the Braun Multiplier.

The resource utilization summary in Figure 18 shows the design used only 91 LUTs and 32 IOs, indicating a low-resource footprint.

Summary

Resource	Utilization	Available	Utilization %
LUT	87	20800	0.42
IO	32	106	30.19

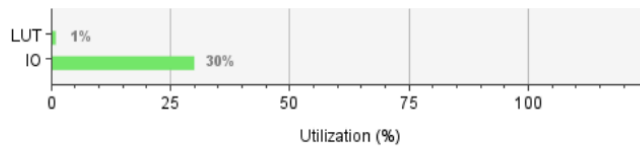


Figure 18 Brauns Resource utilization summary.

Figure 19 outlines the pin allocation for the multipliers.

All ports (32)				
A (8)				
A[7]	IN		R2	✓
A[6]	IN		T1	✓
A[5]	IN		U1	✓
A[4]	IN		W2	✓
A[3]	IN		R3	✓
A[2]	IN		T2	✓
A[1]	IN		T3	✓
A[0]	IN		V2	✓
B (8)				
B[7]	IN		W13	✓
B[6]	IN		W14	✓
B[5]	IN		V15	✓
B[4]	IN		W15	✓
B[3]	IN		W17	✓
B[2]	IN		W16	✓
B[1]	IN		V16	✓
B[0]	IN		V17	✓
P (16)				
P[15]	OUT		L1	✓
P[14]	OUT		P1	✓
P[13]	OUT		N3	✓
P[12]	OUT		P3	✓
P[11]	OUT		U3	✓
P[10]	OUT		W3	✓
P[9]	OUT		V3	✓
P[8]	OUT		V13	✓
P[7]	OUT		V14	✓
P[6]	OUT		U14	✓
P[5]	OUT		U15	✓
P[4]	OUT		W18	✓
P[3]	OUT		V19	✓
P[2]	OUT		U19	✓
P[1]	OUT		E19	✓
P[0]	OUT		U16	✓
Scalar ports (0)				

Figure 19 Pin allocation mapping inputs to switches and outputs to LEDs

Figure 20 shows the Braun Multiplier running live on hardware, with LEDs confirming output for FF×FF.

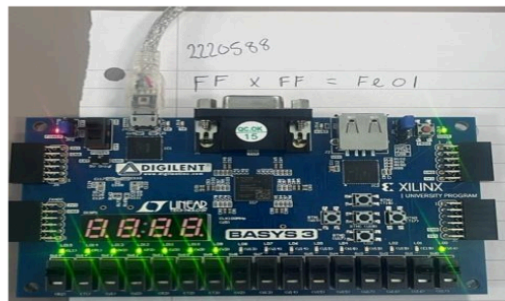


Figure 20 Braun Multiplier deployed on Basys 3 board with live output shown via LEDs (FF×FF).

The Wallace Tree multiplier was also implemented on the Basys 3 board. Due to its parallel architecture, internal logic blocks are more spread out. The Vivado floorplan for Wallace is shown in Figure 21, and the routing layout is shown in Figure 22.

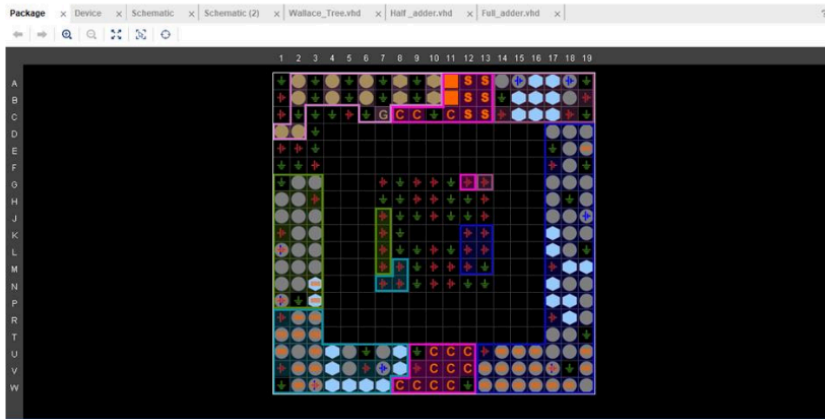


Figure 21 Vivado floorplan for the Wallace Tree

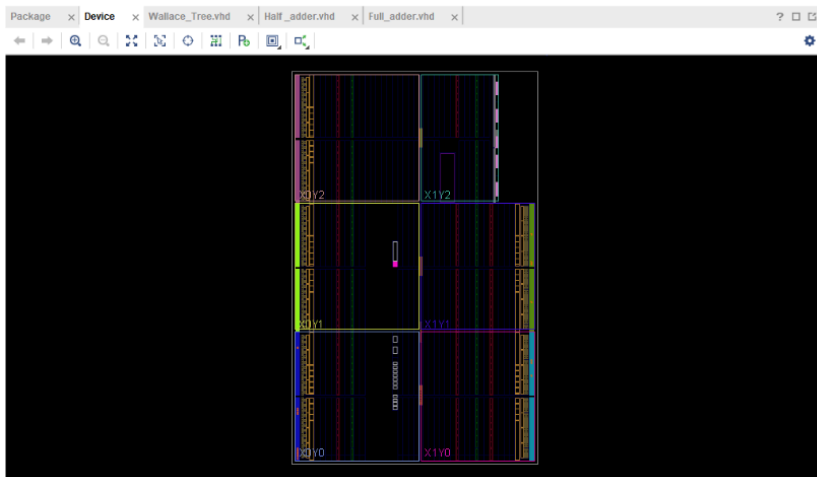


Figure 22 FPGA routing layout showing signal interconnections for the Braun Multiplier.

Figure 23 confirms similar resource usage to Braun (91 LUTs, 32 IOs), demonstrating that Wallace maintains efficiency despite higher complexity. Figure 24 shows the Wallace Tree running on hardware, displaying a live output from test case 08x05.

Summary

Resource	Utilization	Available	Utilization %
LUT	91	20800	0.44
IO	32	106	30.19

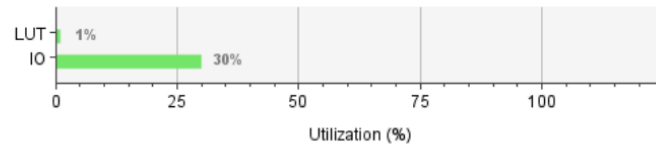


Figure 23 Wallace Tree Resource Utilization Summary.

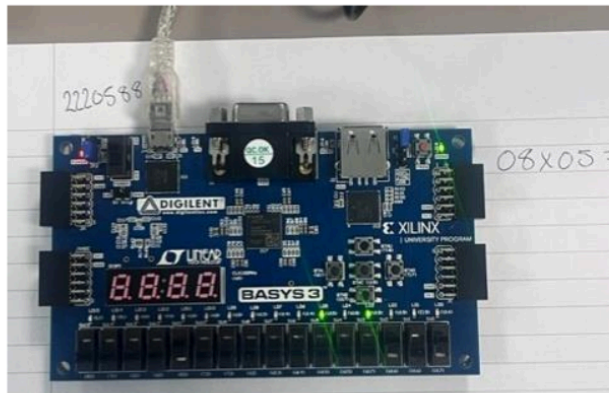


Figure 24 Wallace Tree Multiplier deployed on Basys 3 board with live output shown via LEDs (08x05).

Figures 25 and 26 show the RTL schematics post-synthesis for Braun and Wallace Tree, generated from their VHDL source. Braun's structure is regular and grid-like, while the Wallace design is less uniform due to compression layers.

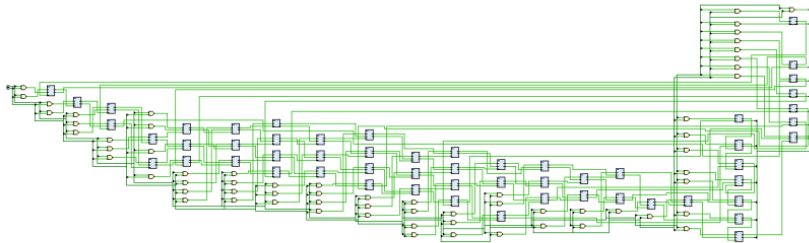


Figure 25 RTL schematic of Braun multiplier post-synthesis

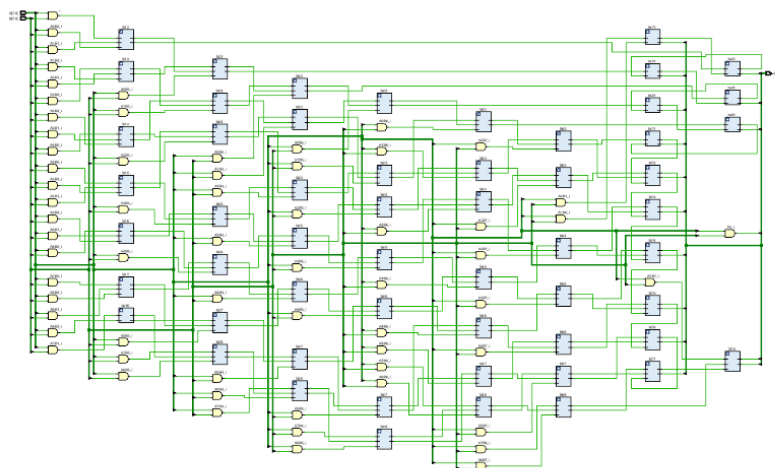


Figure 26 RTL schematic of Wallace Tree multiplier post-synthesis

Figure 27-41 presents Vivado's detailed delay report for the Braun multiplier. The report confirms the highest propagation delays occur from input A to output P, peaking above 21 ns consistent with the manual critical path estimate and explaining Braun's slower timing performance [11].

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[0]	P[0]	8.940	SLOW	2.717	FAST
A[0]	P[1]	8.610	SLOW	2.616	FAST
A[0]	P[2]	8.753	SLOW	2.667	FAST
A[0]	P[3]	9.557	SLOW	2.756	FAST
A[0]	P[4]	11.487	SLOW	2.665	FAST
A[0]	P[5]	13.089	SLOW	2.618	FAST
A[0]	P[6]	13.979	SLOW	2.643	FAST
A[0]	P[7]	15.235	SLOW	2.788	FAST
A[0]	P[8]	17.494	SLOW	3.217	FAST
A[0]	P[9]	19.777	SLOW	3.754	FAST
A[0]	P[10]	19.859	SLOW	3.709	FAST
A[0]	P[11]	20.684	SLOW	4.069	FAST
A[0]	P[12]	21.519	SLOW	4.330	FAST
A[0]	P[13]	21.110	SLOW	4.238	FAST
A[0]	P[14]	21.550	SLOW	4.486	FAST
A[0]	P[15]	21.561	SLOW	4.493	FAST

Figure 27 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[1]	P[1]	9.397	SLOW	2.876	FAST
A[1]	P[2]	9.543	SLOW	2.926	FAST
A[1]	P[3]	9.715	SLOW	2.953	FAST
A[1]	P[4]	11.795	SLOW	2.912	FAST
A[1]	P[5]	13.398	SLOW	2.944	FAST
A[1]	P[6]	14.287	SLOW	2.884	FAST
A[1]	P[7]	15.543	SLOW	2.956	FAST
A[1]	P[8]	17.803	SLOW	2.947	FAST
A[1]	P[9]	20.085	SLOW	3.591	FAST
A[1]	P[10]	20.167	SLOW	3.546	FAST
A[1]	P[11]	20.992	SLOW	3.906	FAST
A[1]	P[12]	21.828	SLOW	4.116	FAST
A[1]	P[13]	21.419	SLOW	4.075	FAST
A[1]	P[14]	21.859	SLOW	4.323	FAST
A[1]	P[15]	21.870	SLOW	4.295	FAST

Figure 28 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

A[2]	P[2]	9.059	SLOW	2.739	FAST
A[2]	P[3]	9.371	SLOW	2.808	FAST
A[2]	P[4]	11.486	SLOW	2.927	FAST
A[2]	P[5]	13.088	SLOW	2.954	FAST
A[2]	P[6]	13.977	SLOW	2.919	FAST
A[2]	P[7]	15.233	SLOW	2.894	FAST
A[2]	P[8]	17.493	SLOW	3.001	FAST
A[2]	P[9]	19.775	SLOW	2.932	FAST
A[2]	P[10]	19.857	SLOW	3.148	FAST
A[2]	P[11]	20.683	SLOW	3.508	FAST
A[2]	P[12]	21.518	SLOW	3.768	FAST
A[2]	P[13]	21.109	SLOW	3.676	FAST
A[2]	P[14]	21.549	SLOW	3.924	FAST
A[2]	P[15]	21.560	SLOW	3.931	FAST

Figure 29 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[3]	P[4]	11.545	SLOW	3.066	FAST
A[3]	P[5]	12.813	SLOW	3.279	FAST
A[3]	P[6]	13.702	SLOW	3.154	FAST
A[3]	P[7]	14.958	SLOW	3.086	FAST
A[3]	P[8]	17.218	SLOW	3.220	FAST
A[3]	P[9]	19.500	SLOW	3.316	FAST
A[3]	P[10]	19.583	SLOW	2.837	FAST
A[3]	P[11]	20.408	SLOW	3.299	FAST
A[3]	P[12]	21.243	SLOW	3.252	FAST
A[3]	P[13]	20.834	SLOW	3.388	FAST
A[3]	P[14]	21.274	SLOW	3.489	FAST
A[3]	P[15]	21.285	SLOW	3.431	FAST

Figure 30 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

A[4]	P[4]	10.544	SLOW	3.293	FAST
A[4]	P[5]	11.711	SLOW	3.318	FAST
A[4]	P[6]	12.632	SLOW	3.157	FAST
A[4]	P[7]	13.890	SLOW	3.414	FAST
A[4]	P[8]	16.150	SLOW	3.583	FAST
A[4]	P[9]	18.432	SLOW	3.633	FAST
A[4]	P[10]	18.514	SLOW	3.038	FAST
A[4]	P[11]	19.339	SLOW	2.967	FAST
A[4]	P[12]	20.175	SLOW	3.152	FAST
A[4]	P[13]	19.766	SLOW	3.288	FAST
A[4]	P[14]	20.206	SLOW	3.390	FAST
A[4]	P[15]	20.217	SLOW	3.332	FAST

Figure 31 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

► A[5]	◄ P[5]	11.491	SLOW	3.684	FAST
► A[5]	◄ P[6]	12.380	SLOW	3.450	FAST
► A[5]	◄ P[7]	13.636	SLOW	3.634	FAST
► A[5]	◄ P[8]	16.043	SLOW	3.672	FAST
► A[5]	◄ P[9]	18.325	SLOW	3.893	FAST
► A[5]	◄ P[10]	18.408	SLOW	3.261	FAST
► A[5]	◄ P[11]	19.233	SLOW	3.134	FAST
► A[5]	◄ P[12]	20.068	SLOW	3.051	FAST
► A[5]	◄ P[13]	19.659	SLOW	3.355	FAST
► A[5]	◄ P[14]	20.099	SLOW	3.484	FAST
► A[5]	◄ P[15]	20.110	SLOW	3.425	FAST

Figure 32 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

► A[6]	◄ P[6]	11.107	SLOW	3.562	FAST
► A[6]	◄ P[7]	12.078	SLOW	3.674	FAST
► A[6]	◄ P[8]	14.830	SLOW	3.992	FAST
► A[6]	◄ P[9]	17.113	SLOW	4.213	FAST
► A[6]	◄ P[10]	17.195	SLOW	3.382	FAST
► A[6]	◄ P[11]	18.020	SLOW	3.271	FAST
► A[6]	◄ P[12]	18.855	SLOW	3.137	FAST
► A[6]	◄ P[13]	18.446	SLOW	2.895	FAST
► A[6]	◄ P[14]	18.886	SLOW	2.788	FAST
► A[6]	◄ P[15]	18.898	SLOW	2.733	FAST

Figure 33 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

► A[7]	◄ P[7]	11.541	SLOW	3.587	FAST
► A[7]	◄ P[8]	13.989	SLOW	3.935	FAST
► A[7]	◄ P[9]	16.271	SLOW	4.290	FAST
► A[7]	◄ P[10]	16.353	SLOW	3.593	FAST
► A[7]	◄ P[11]	17.179	SLOW	3.550	FAST
► A[7]	◄ P[12]	18.014	SLOW	3.112	FAST
► A[7]	◄ P[13]	17.605	SLOW	2.894	FAST
► A[7]	◄ P[14]	18.045	SLOW	2.749	FAST
► A[7]	◄ P[15]	18.056	SLOW	2.698	FAST

Figure 34 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
B[0]	P[0]	9.098	SLOW	2.771	FAST
B[0]	P[1]	8.801	SLOW	2.669	FAST
B[0]	P[2]	8.636	SLOW	2.617	FAST
B[0]	P[3]	9.759	SLOW	2.823	FAST
B[0]	P[4]	11.689	SLOW	3.111	FAST
B[0]	P[5]	13.291	SLOW	3.170	FAST
B[0]	P[6]	14.181	SLOW	3.237	FAST
B[0]	P[7]	15.436	SLOW	3.550	FAST
B[0]	P[8]	17.696	SLOW	3.818	FAST
B[0]	P[9]	19.979	SLOW	4.274	FAST
B[0]	P[10]	20.061	SLOW	3.987	FAST
B[0]	P[11]	20.886	SLOW	3.944	FAST
B[0]	P[12]	21.721	SLOW	3.941	FAST
B[0]	P[13]	21.312	SLOW	3.910	FAST
B[0]	P[14]	21.752	SLOW	4.012	FAST
B[0]	P[15]	21.763	SLOW	3.953	FAST

Figure 35 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

B[1]	P[1]	8.855	SLOW	2.650	FAST
B[1]	P[2]	9.040	SLOW	2.710	FAST
B[1]	P[3]	9.366	SLOW	2.667	FAST
B[1]	P[4]	11.296	SLOW	2.955	FAST
B[1]	P[5]	12.898	SLOW	3.168	FAST
B[1]	P[6]	13.787	SLOW	3.272	FAST
B[1]	P[7]	15.043	SLOW	3.582	FAST
B[1]	P[8]	17.303	SLOW	3.853	FAST
B[1]	P[9]	19.586	SLOW	4.335	FAST
B[1]	P[10]	19.668	SLOW	3.822	FAST
B[1]	P[11]	20.493	SLOW	3.779	FAST
B[1]	P[12]	21.328	SLOW	3.776	FAST
B[1]	P[13]	20.919	SLOW	3.745	FAST
B[1]	P[14]	21.359	SLOW	3.846	FAST
B[1]	P[15]	21.370	SLOW	3.788	FAST

Figure 36 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

► B[2]	◄ P[2]	8.927	SLOW	2.703	FAST
► B[2]	◄ P[3]	10.045	SLOW	2.767	FAST
► B[2]	◄ P[4]	11.975	SLOW	2.958	FAST
► B[2]	◄ P[5]	13.577	SLOW	3.154	FAST
► B[2]	◄ P[6]	14.466	SLOW	3.019	FAST
► B[2]	◄ P[7]	15.722	SLOW	3.289	FAST
► B[2]	◄ P[8]	17.982	SLOW	3.569	FAST
► B[2]	◄ P[9]	20.265	SLOW	4.107	FAST
► B[2]	◄ P[10]	20.347	SLOW	3.618	FAST
► B[2]	◄ P[11]	21.172	SLOW	3.575	FAST
► B[2]	◄ P[12]	22.007	SLOW	3.572	FAST
► B[2]	◄ P[13]	21.598	SLOW	3.541	FAST
► B[2]	◄ P[14]	22.038	SLOW	3.643	FAST
► B[2]	◄ P[15]	22.049	SLOW	3.584	FAST

Figure 37 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

B[3]	P[3]	8.494	SLOW	2.552	FAST
B[3]	P[4]	9.530	SLOW	2.540	FAST
B[3]	P[5]	11.561	SLOW	2.913	FAST
B[3]	P[6]	12.481	SLOW	2.910	FAST
B[3]	P[7]	13.740	SLOW	2.873	FAST
B[3]	P[8]	16.000	SLOW	3.221	FAST
B[3]	P[9]	18.282	SLOW	3.720	FAST
B[3]	P[10]	18.364	SLOW	3.651	FAST
B[3]	P[11]	19.189	SLOW	3.670	FAST
B[3]	P[12]	20.025	SLOW	3.667	FAST
B[3]	P[13]	19.615	SLOW	3.636	FAST
B[3]	P[14]	20.056	SLOW	3.737	FAST
B[3]	P[15]	20.067	SLOW	3.679	FAST

Figure 38 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

B[4]	P[4]	8.176	SLOW	2.402	FAST
B[4]	P[5]	9.209	SLOW	2.399	FAST
B[4]	P[6]	10.971	SLOW	2.564	FAST
B[4]	P[7]	12.272	SLOW	2.846	FAST
B[4]	P[8]	14.371	SLOW	3.114	FAST
B[4]	P[9]	16.651	SLOW	3.601	FAST
B[4]	P[10]	16.733	SLOW	3.212	FAST
B[4]	P[11]	17.558	SLOW	3.170	FAST
B[4]	P[12]	18.394	SLOW	3.234	FAST
B[4]	P[13]	17.984	SLOW	3.202	FAST
B[4]	P[14]	18.425	SLOW	3.304	FAST
B[4]	P[15]	18.436	SLOW	3.246	FAST

Figure 39 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

B[5]	P[5]	8.338	SLOW	2.423	FAST
B[5]	P[6]	10.170	SLOW	2.704	FAST
B[5]	P[7]	11.501	SLOW	2.664	FAST
B[5]	P[8]	13.600	SLOW	3.012	FAST
B[5]	P[9]	15.880	SLOW	3.327	FAST
B[5]	P[10]	15.962	SLOW	3.209	FAST
B[5]	P[11]	16.787	SLOW	3.080	FAST
B[5]	P[12]	17.623	SLOW	3.108	FAST
B[5]	P[13]	17.214	SLOW	3.046	FAST
B[5]	P[14]	17.654	SLOW	3.178	FAST
B[5]	P[15]	17.665	SLOW	3.119	FAST

Figure 40 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

B[6]	P[6]	8.383	SLOW	2.445	FAST
B[6]	P[7]	8.966	SLOW	2.472	FAST
B[6]	P[8]	10.731	SLOW	2.674	FAST
B[6]	P[9]	13.011	SLOW	2.970	FAST
B[6]	P[10]	13.093	SLOW	2.865	FAST
B[6]	P[11]	13.918	SLOW	2.676	FAST
B[6]	P[12]	14.754	SLOW	2.728	FAST
B[6]	P[13]	14.345	SLOW	2.864	FAST
B[6]	P[14]	14.785	SLOW	2.966	FAST
B[6]	P[15]	14.796	SLOW	2.868	FAST
B[7]	P[7]	8.571	SLOW	2.560	FAST
B[7]	P[8]	9.276	SLOW	2.375	FAST
B[7]	P[9]	10.733	SLOW	2.576	FAST
B[7]	P[10]	10.754	SLOW	2.573	FAST
B[7]	P[11]	11.773	SLOW	2.539	FAST
B[7]	P[12]	12.609	SLOW	2.462	FAST
B[7]	P[13]	12.199	SLOW	2.514	FAST
B[7]	P[14]	12.640	SLOW	2.618	FAST
B[7]	P[15]	12.651	SLOW	2.563	FAST

Figure 41 Detailed timing delay report for the Braun multiplier showing maximum and minimum propagation delays across all output ports (Vivado)

Simulation Results:

Functional simulations were carried out in Vivado to verify the correct functionality of both the Braun and Wallace Tree multipliers. Identical input values were applied to both designs to ensure a fair and consistent comparison.

Each design was tested using a range of hexadecimal input pairs, covering low, mid, and high values to ensure full functionality:

1. 05 x 08 – Low range
2. AB x BA – Mid-range
3. AC x DC – Mid-range
4. FF x FF – High range

Waveform outputs were captured for each test case, and every result was verified against the expected binary and hexadecimal products.

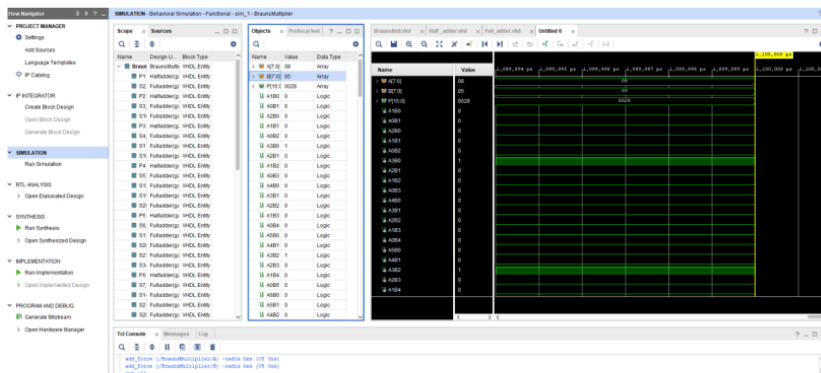


Figure 42 Braun8x8 Multiplier Lowest Value 08 x 05 = 0028



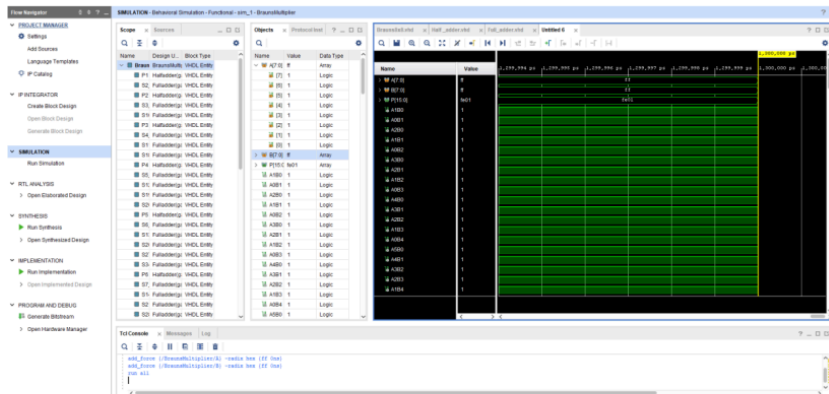


Figure 45 Braun8x8 Multiplier High Range value Braun FF x FF = fe01

Following on with Wallace Tree Multiplier Results.

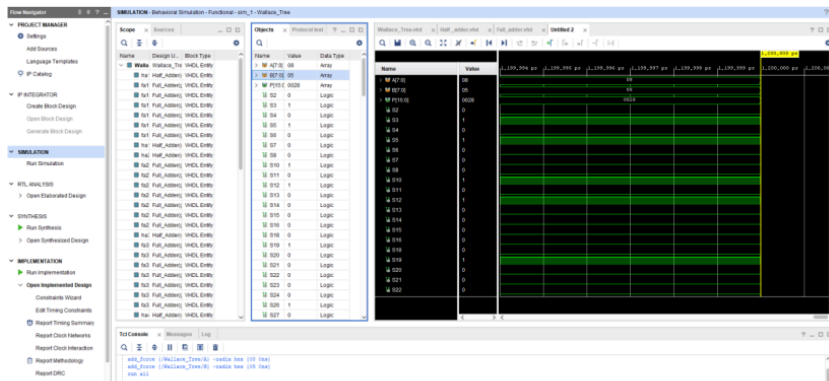


Figure 46 Wallace tree Multiplier Lowest Value 08 x 05 = 0028

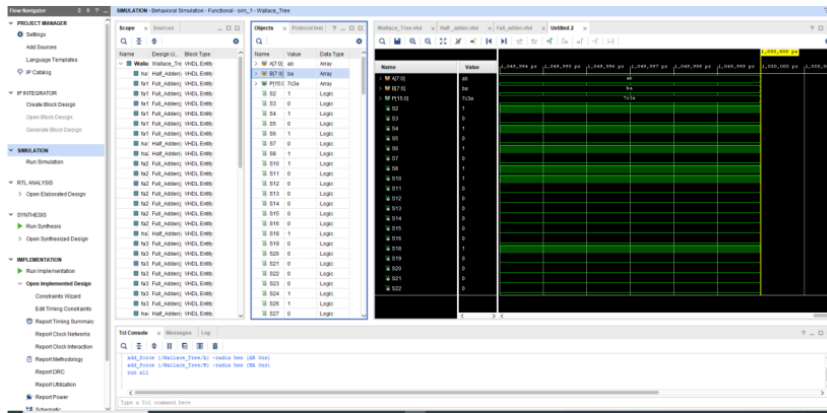


Figure 47 Wallace tree Multiplier Mid Valve 1 $AB \times BA = 7c3e$

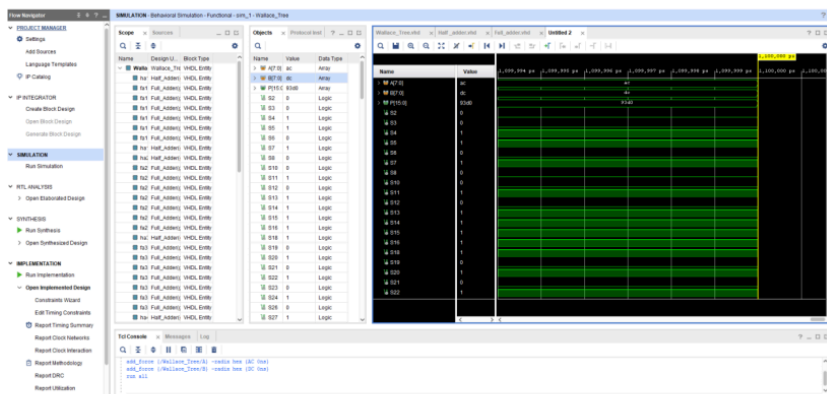


Figure 48 Wallace tree Multiplier Mid Valve 2 $AC \times DC = 93d0$

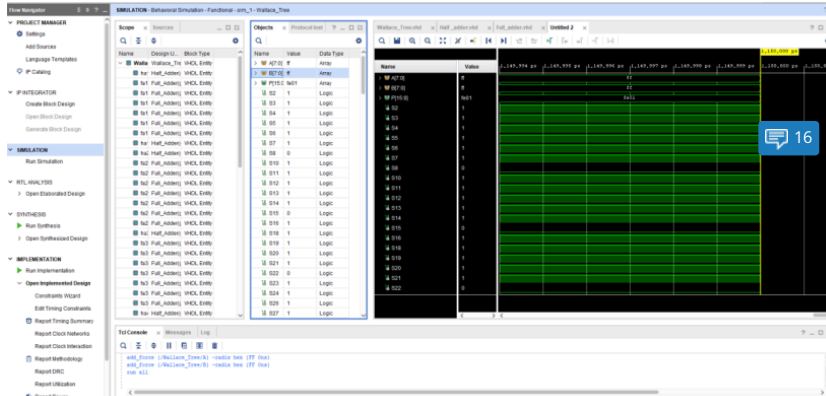


Figure 49 Wallace tree Multiplier High Value ff x ff = fe01

To summarise the simulation outcomes, the table below shows each test along with the output status for both multipliers. All outputs were verified against the expected result.

Sample	Input A (Hex)	Input B (Hex)	Output (Hex)	Braun Result	Wallace Tree Result
Low	08	05	0028	Pass	Pass
Mid 1	AB	BA	7C3E	Pass	Pass
Mid 2	AC	BA	93D0	Pass	Pass
High	FF	FF	FE01	Pass	Pass

Table 3 Results of Simulation testing for both multipliers

Comparison, analysis & Interpretation of results:

This section compares both multiplier designs — Braun and Wallace Tree — based on simulation, manual timing calculations, Vivado synthesis reports, and hardware testing. The aim is to see which one performs better overall and which is better suited for specific use cases.

Time Analysis

Manually, the Braun multiplier has a longer critical path due to its sequential adder structure:

$$1 \text{ ns} + (14 \times 2 \text{ ns}) = \mathbf{29 \text{ ns}}$$

The Wallace Tree compresses partial products using a parallel structure, reducing logic levels:

$$1 \text{ ns} + (7 \times 2 \text{ ns}) = \mathbf{15 \text{ ns}}$$

Vivado synthesis results supported these estimates:

Metric	Braun	Wallace Tree
Estimated Delay	29 ns	15 ns
Vivado WNS	13.5 ns	6.2 ns
Timing Closure	Harder	Easier
Critical Path Type	Sequential adders	Tree compression

Table 4 Timing Evaluation

Wallace Tree clearly performs better in terms of speed, making it ideal for high-frequency or performance-critical systems.

Test Accuracy and Functionality

Both designs passed all four test samples on hardware using the same pin configuration and testbench. Simulation waveforms and LED outputs matched across all cases:

1. 08 × 05
2. AB × BA
3. AC × DC
4. FF × FF

This confirms both architectures were functionally accurate in Vivado and in real FPGA operation.

Design Efficiency and Use Case Comparison

Although Wallace Tree is faster, the Braun multiplier used slightly fewer resources and consumed marginally less power making it more suitable for compact, low-power applications [13].

Metric	Braun	Wallace Tree
LUTs Used	87	91
I/O Pins Used	32	32
Total Power	0.080 W	0.084 W
Design Type	Simple & compact	Fast & parallel
Best For	Low-power apps	High-speed logic

Table 5 LUT Comparison

These numbers reflect findings from [13], which also showed Wallace Tree structures offering better delay performance at a slight cost in logic area especially relevant in high-performance FPGA systems.

Interpretation of Power Utilization and LUTs:

To assess how each multiplier design performs in terms of FPGA resource usage and power efficiency, Vivado was used to generate detailed reports covering power breakdown and logic resource LUT utilization. Key comparisons include total power, static vs dynamic distribution, and LUT demand.

Braun Multiplier – Power Breakdown

Vivado reported a total power of **0.080 W** for the Braun multiplier. As shown in **Figure 50**, the majority (89%) was **static power**, with minimal dynamic switching activity. Only **0.008 W** was used for I/O, suggesting efficient internal logic and minimal toggling.

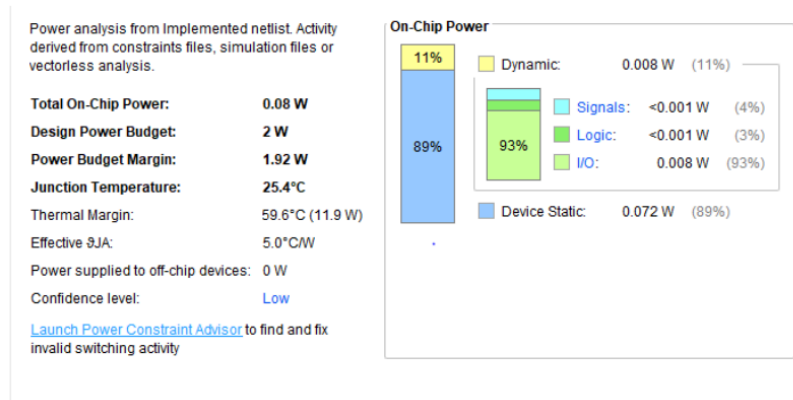


Figure 50 Power Report for Braun Multiplier

The Braun design is quite low power overall, with 0.008 W used for I/O. This shows that it's not doing any excessive toggling or complex switching logic internally.

Braun Multiplier – LUT Usage

Figure 51 shows Braun used just **87 LUTs**, which represents **0.42%** of the available 20,800 on the Artix-7 FPGA. I/O usage was higher at **30%**, due to 16 inputs and 16 outputs being mapped to switches and LEDs.

Summary

Resource	Utilization	Available	Utilization %
LUT	87	20800	0.42
IO	32	106	30.19

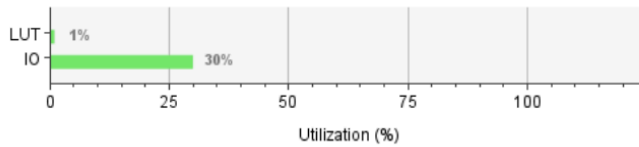


Figure 51 Braun Utilization LUT

Overall, Braun is compact and resource-light ideal for low-power or constrained environments.

Wallace Tree Multiplier – Power Breakdown

The Wallace Tree design consumed 0.084W as shown in figure 52, slightly higher than Braun, with I/O power rising to 0.011 W. This is expected due to its parallel reduction stages, which increase switching activity and fan-out across deeper logic layers. The architecture's increased toggle rate leads to higher dynamic power consumption, although it remains well within FPGA limits.

Despite the rise in dynamic power, the performance benefits of Wallace Tree, especially in terms of timing, make it ideal for high-speed applications. Future iterations could consider clock gating or operand isolation to reduce unnecessary toggling and further optimize energy usage [12].

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power: 0.084 W
Design Power Budget: 2 W
Power Budget Margin: 1.916 W
Junction Temperature: 25.4°C
 Thermal Margin: 59.6°C (11.9 W)
 Effective θ_{JA} : 5.0°C/W
 Power supplied to off-chip devices: 0 W
 Confidence level: Low

[Launch Power Constraint Advisor](#) to find and fix invalid switching activity

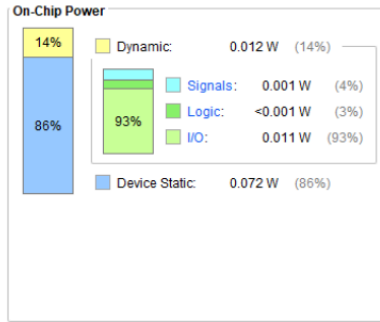


Figure 52 Power Report Wallace Tree Multiplier

Despite the increase, the Wallace design remains efficient, staying well under the 2W FPGA limit.

Wallace Tree – LUT Usage

Summary

Resource	Utilization	Available	Utilization %
LUT	91	20800	0.44
IO	32	106	30.19

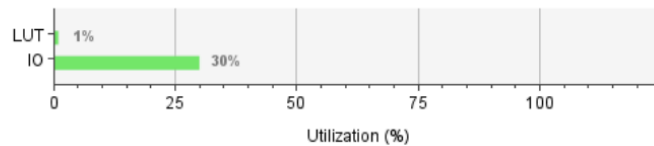


Figure 53 Wallace Tree Resource utilization summary.

As seen in Figure 53, Wallace used 91 LUTs (0.44%), which is just 4 more than Braun. I/O usage remained the same at 32 pins (30.19%). The slightly higher LUT count supports the faster performance shown in previous timing analysis. Side by side comparison table 5.

Metric	Braun	Wallace Tree
--------	-------	--------------

Total Power (W)	0.080	0.084
I/O Power (W)	0.008	0.011
Static Power (W)	0.072	0.072
LUTs Used	87	91
LUT Utilisation %	0.42%	0.44%
IO Used	32	32
IO Utilisation %	30.19%	30.19%

Table 6 Power Report Comparison

From this, you can clearly see that both designs are light in terms of power and logic, but the Braun edges ahead slightly for efficiency, while Wallace gives you that performance boost at the cost of just a bit more resources further displayed in figure 54 below.

Name	Slice LUTs (20800)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)
N BraunsMultiplier	87	27	87	32

Name	Slice LUTs (20800)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)
N Wallace_Tree	91	30	91	32

Figure 54 side by side comparison

Both designs use FPGA resources efficiently and meet power constraints. Braun is better for simplicity and low-power use, ideal in embedded systems. Wallace Tree, while slightly more power-hungry, offers superior timing thanks to its parallel architecture. This tradeoff is justified in high-performance scenarios where speed matters more than minor increases in logic or power use [12].

Details of testing and future improvements:

Hardware testing was conducted following simulation to verify the functionality of both multipliers. A range of 8-bit binary input combinations were used during simulation in Vivado, and the same values were applied in real-time on the Basys 3 FPGA board. The test conditions were kept identical across both designs.

Outputs were observed on the onboard LEDs, and each binary result matched the expected product confirming that the logic was correct across both simulation and hardware.

Figure 55 below displays a collection of hardware testing results, showing successful functionality for both architectures on the physical FPGA board

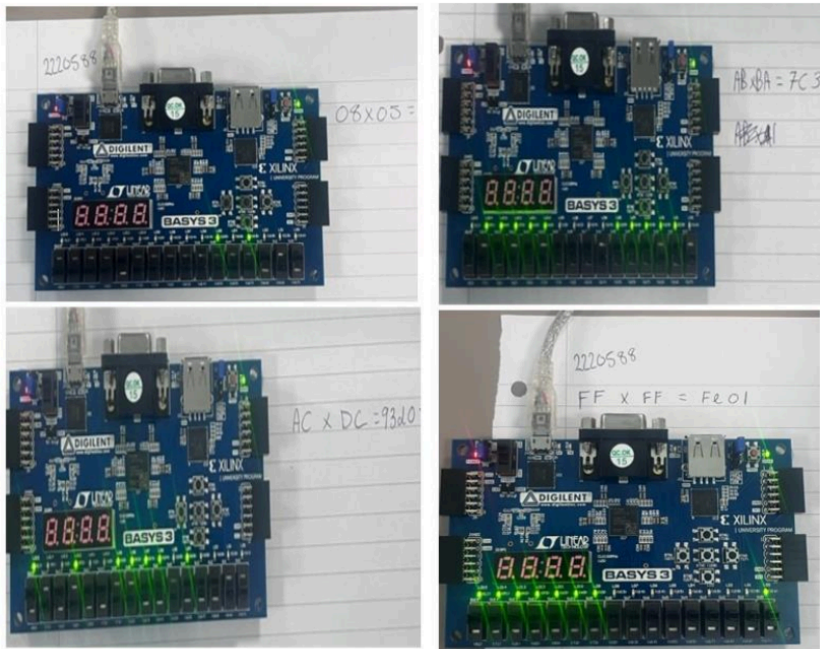


Figure 55 Hardware test outputs from the Basys 3 board for multiple 8x8 input combinations

Future considerations

- **Pipelining**

Implementing pipelining in the Wallace Tree could reduce its critical path delay further, allowing operation at higher clock frequencies. This would improve throughput, especially for larger operand sizes or repetitive operations.

- **Power Reduction**

Methods like clock gating or operand gating can lower dynamic power, which is essential for low-energy or embedded designs.

- **Signed Operation Support**

Adding support for signed multiplication (e.g., 2's complement) would allow broader use in DSP, image processing, and control systems [13].

- **Scalability**

As operand width increases, the regularity of the Braun multiplier simplifies design scaling, though at the cost of deeper adder chains. In contrast, the Wallace Tree's complexity grows with additional stages, making routing and timing closure more difficult.

- **ALU Integration**

These multipliers could be embedded within a full Arithmetic Logic Unit (ALU) or integrated as part of a processor Design [14].

The trade-off Comparison of Braun and Wallace Tree Multipliers is shown in Table 7.

Feature	Braun	Wallace Tree
Delay	Higher	Lower
Power Efficiency	Better	Slightly worse
FPGA Resource Usage	Lower	Slightly higher
Scalability (Size)	Easier to scale	Complex to scale
Pipelining Support	Limited benefit	High benefit

Table 7 Comparison of Braun and Wallace Tree Multipliers

Conclusion:

Both Braun and Wallace Tree multipliers were successfully implemented, simulated, and tested on FPGA hardware. The Braun design offers simplicity, lower power, and efficient resource usage, making it suitable for embedded or low-power applications. Wallace Tree, though more complex, delivers significantly better timing performance ideal for speed-critical systems.

The project highlighted how architectural design directly impacts power, timing, and hardware utilization in FPGA workflows. Future work could explore signed multiplication, pipelined versions for higher frequency operation, or integration into full ALUs. Ultimately, the choice between these designs depends on system goals Braun for efficiency, Wallace for performance. [11].

References:

- [1] R. B. Rani and P. Sivakumar, "Design and analysis of Braun and Wallace Tree multipliers using VHDL," *Int. J. Sci. Res.*, vol. 5, no. 3, pp. 112–115, Mar. 2016.
- [2] B. Braun, "A digit-serial multiplication algorithm for parallel multipliers," *IEEE Trans. Comput.*, vol. C-22, no. 2, pp. 118–120, 1973.
- [3] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann, 2013.
- [4] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-13, no. 1, pp. 14–17, Feb. 1964.
- [5] A. C. Swathi and T. Yuvraj, "High-performance Wallace Tree multiplier," *Int. J. Comput. Tech.*, vol. 7, no. 1, pp. 5–8, Feb. 2020. [Online]. Available: https://www.researchgate.net/publication/343962319_High-Performance_Wallace_Tree_Multiplier-
- [6] M. M. Mano and M. D. Ciletti, *Digital Design*, 5th ed. Harlow, U.K.: Pearson, 2013.
- [7] C. H. Roth Jr. and L. L. Kinney, *Fundamentals of Logic Design*, 7th ed. Boston, MA, USA: Cengage Learning, 2013.
- [8] D. Harris and S. Harris, *Digital Design and Computer Architecture*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, 2012.
- [9] J. Bhasker, *A VHDL Primer*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 1999.
- [10] D. Lakshmaiah, M. V. Subramanyam, and K. S. Prasad, "Design of low power 4-bit CMOS Braun multiplier based on threshold voltage techniques," *Glob. J. Res. Eng.: F Electr. Electron. Eng.*, vol. 14, no. 9, pp. 1–6, 2014.
- [11] W. J. Townsend, E. E. Swartzlander, and J. A. Abraham, "A comparison of Dadda and Wallace multiplier delays," in *Adv. Signal Process. Algorithms, Archit., and Implementations XIII*, vol. 5205, SPIE, 2003, pp. 552–560.
- [12] J. Anderson and Q. Wang, "Power-aware FPGA design: Challenges and opportunities," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 30, no. 8, pp. 1176–1189, Aug. 2011.
- [13] S. Singh, A. Yadav, and K. Rana, "Performance analysis of Wallace, Dadda and Braun multipliers on FPGA," *Int. J. Eng. Adv. Technol.*, vol. 8, no. 6, pp. 1238–1242, Aug. 2019.

[14] M. D. Ciletti, *Advanced Digital Design with the Verilog HDL*, 2nd ed. Boston, MA, USA: Pearson, 2011.



Appendix

Brauns Code

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  -- Uncomment the following library declaration if using
4  -- arithmetic functions with Signed or Unsigned values
5  --use IEEE.NUMERIC_STD.ALL;
6
7  -- Uncomment the following library declaration if instantiating
8  -- any Xilinx leaf cells in this code.
9  --library UNISIM;
10 --use UNISIM.VComponents.all;
11
12
13 entity BraunsMultiplier is
14     Port ( A : in std_logic_vector (7 downto 0);
15           B : in std_logic_vector (7 downto 0);
16           P : out std_logic_vector (15 downto 0));
17
18 end BraunsMultiplier;
19
20 architecture Behavioral of BraunsMultiplier is
21     signal A1B0 : std_logic;
22     signal A0B1 : std_logic;
23     signal A2B0 : std_logic;
24     signal A1B1 : std_logic;
25     signal A0B2 : std_logic;
26     signal A3B0 : std_logic;
27     signal A2B1 : std_logic;
28     signal A1B2 : std_logic;
29     signal A0B3 : std_logic;
```

Figure 56 Braun Code

```

86 : signal C1 : std_logic;--P2
87 : signal S2 : std_logic;
88 :
89 : signal C2 : std_logic;--P3
90 : signal S3 : std_logic;
91 : signal C9 : std_logic;
92 : signal S10 : std_logic;
93 :
94 : signal C3 : std_logic;--P4
95 : signal S4 : std_logic;
96 : signal C10 : std_logic;
97 : signal S11 : std_logic;
98 : signal C17 : std_logic;
99 : signal S18 : std_logic;
.00 :
.01 : signal C4 : std_logic;--P5
.02 : signal S5 : std_logic;
.03 : signal C11 : std_logic;
.04 : signal S12 : std_logic;
.05 : signal C18 : std_logic;
.06 : signal S19 : std_logic;
.07 : signal C25 : std_logic;
.08 : signal S26 : std_logic;
.09 :
.10 : signal C5 : std_logic;--P6
.11 : signal S6 : std_logic;
.12 : signal C12 : std_logic;
.13 : signal S13 : std_logic;
.14 : signal C19 : std_logic;
.15 : signal S20 : std_logic;
.16 : signal C26 : std_logic;
.17 : signal S27 : std_logic;
.18 : signal C33 : std_logic;
.19 : signal S34 : std_logic;
.20 :
.21 : signal C6 : std_logic;--P7
.22 : signal S7 : std_logic;
.23 : signal C13 : std_logic;
.24 : signal S14 : std_logic;
.25 : signal C20 : std_logic;
.26 : signal S21 : std_logic;
.27 : signal C27 : std_logic;
.28 : signal S28 : std_logic;
.29 : signal C34 : std_logic;
-- :

```

Figure 57 Braun code

```

216 ○ A1B0 <= A(1) and B(0);
217 ○ A0B1 <= A(0) and B(1);
218 ○ A2B0 <= A(2) and B(0);
219 ○ A1B1 <= A(1) and B(1);
220 ○ A0B2 <= A(0) and B(2);
221 ○ A3B0 <= A(3) and B(0);
222 ○ A2B1 <= A(2) and B(1);
223 ○ A1B2 <= A(1) and B(2);
224 ○ A0B3 <= A(0) and B(3);
225 ○ A4B0 <= A(4) and B(0);
226 ○ A3B1 <= A(3) and B(1);
227 ○ A2B2 <= A(2) and B(2);
228 ○ A1B3 <= A(1) and B(3);
229 ○ A0B4 <= A(0) and B(4);
230 ○ A5B0 <= A(5) and B(0);
231 ○ A4B1 <= A(4) and B(1);
232 ○ A3B2 <= A(3) and B(2);
233 ○ A2B3 <= A(2) and B(3);
234 ○ A2B1 <= A(2) and B(1);
235 ○ A1B4 <= A(1) and B(4);
236 ○ A0B5 <= A(0) and B(5);
237 ○ A6B0 <= A(6) and B(0);
238 ○ A5B1 <= A(5) and B(1);
239 ○ A4B2 <= A(4) and B(2);
240 ○ A3B3 <= A(3) and B(3);
241 ○ A2B4 <= A(2) and B(4);
242 ○ A1B5 <= A(1) and B(5);
243 ○ A0B6 <= A(0) and B(6);
244 ○ A7B0 <= A(7) and B(0);
245 ○ A6B1 <= A(6) and B(1);
246 ○ A5B2 <= A(5) and B(2);
247 ○ A4B3 <= A(4) and B(3);
248 ○ A3B4 <= A(3) and B(4);
249 ○ A2B5 <= A(2) and B(5);
250 ○ A1B6 <= A(1) and B(6);
251 ○ A0B7 <= A(0) and B(7);
252 ○ A7B1 <= A(7) and B(1);
253 ○ A6B2 <= A(6) and B(2);
254 ○ A5B3 <= A(5) and B(3);
255 ○ A4B4 <= A(4) and B(4);
256 ○ A3B5 <= A(3) and B(5);
257 ○ A2B6 <= A(2) and B(6);
258 ○ A1B7 <= A(1) and B(7);

```

Figure 58 Braun code

```

281  P0 : F(0) <= A(0) and B(0);--P0
282
283  P1 : Halfadder port map (A=>A1B0,B=>A0B1,S=>P(1),COUT=>C1);--P1
284
285  S2_1 : Fulladder port map (A=>A2B0,B=>A1B1,CIN=>C1,S=>S2,COUT=>C2);--P2
286  P2: Halfadder port map (A=>A0B2 , B=>S2 , S=>P(2) , COUT=>C9);
287
288  S3_1 : Fulladder port map (A=>A3B0 , B=>A2B1 , CIN=>C2 , S=>S3 , COUT=>C3);--P3
289  S10_1 : Fulladder port map (A=>A1B2 , B=>S3 , CIN=>C9 , S=>S10 , COUT=>C10);
290  P3 : Halfadder port map (A=>A0B3 , B=>S10 , S=>P(3) , COUT=>C17);
291
292  S4_1 : Fulladder port map (A=>A4B0 , B=>A3B1 , CIN=>C3 , S=>S4 , COUT=>C4);--P4
293  S11_1 : Fulladder port map (A=>A2B2 , B=>S4 , CIN=>C10 , S=>S11 , COUT=>C11);
294  S18_1 : Fulladder port map (A=>A1B3 , B=>S11 , CIN=>C17 , S=>S18 , COUT=>C18);
295  P4 : Halfadder port map (A=>A0B4 , B=>S18 , S=>P(4) , COUT=>C25);
296
297  S5_1 : Fulladder port map (A=>A5B0 , B=>A4B1 , CIN=>C4 , S=>S5 , COUT=>C5);--P5
298  S12_1 : Fulladder port map (A=>A3B2 , B=>S5 , CIN=>C11 , S=>S12 , COUT=>C12);
299  S19_1 : Fulladder port map (A=>A2B3 , B=>S12 , CIN=>C18 , S=>S19 , COUT=>C19);
300  S26_1 : Fulladder port map (A=>A1B4 , B=>S19 , CIN=>C25 , S=>S26 , COUT=>C26);
301  P5 : Halfadder port map (A=>A0B5 , B=>S26 , S=>P(5) , COUT=>C33);
302
303  S6_1 : Fulladder port map (A=>A6B0 , B=>A5B1 , CIN=>C5 , S=>S6 , COUT=>C6);--P6
304  S13_1 : Fulladder port map (A=>A4B2 , B=>S6 , CIN=>C12 , S=>S13 , COUT=>C13);
305  S20_1 : Fulladder port map (A=>A3B3 , B=>S13 , CIN=>C19 , S=>S20 , COUT=>C20);
306  S27_1 : Fulladder port map (A=>A2B4 , B=>S20 , CIN=>C26 , S=>S27 , COUT=>C27);
307  S34_1 : Fulladder port map (A=>A1B5 , B=>S27 , CIN=>C33 , S=>S34 , COUT=>C34);
308  P6 : Halfadder port map (A=>A0B6 , B=>S34 , S=>P(6) , COUT=>C41);
309
310  S7_1 : Fulladder port map (A=>A7B0 , B=>A6B1 , CIN=>C6 , S=>S7 , COUT=>C7);--P7
311  S14_1 : Fulladder port map (A=>A5B2 , B=>S7 , CIN=>C13 , S=>S14 , COUT=>C14);
312  S21_1 : Fulladder port map (A=>A4B3 , B=>S14 , CIN=>C20 , S=>S21 , COUT=>C21);
313  S28_1 : Fulladder port map (A=>A3B4 , B=>S21 , CIN=>C27 , S=>S28 , COUT=>C28);
314  S35_1 : Fulladder port map (A=>A2B5 , B=>S28 , CIN=>C34 , S=>S35 , COUT=>C35);
315  S42_1 : Fulladder port map (A=>A1B6 , B=>S35 , CIN=>C41 , S=>S42 , COUT=>C42);
316  P7 : Halfadder port map (A=>A0B7 , B=>S42 , S=>P(7) , COUT=>C49);
317
318  S8_1 : Halfadder port map (A=>A7B1 , B=>C7 , S=>S8 , COUT=>C8);--P8
319  S15_1 : Fulladder port map (A=>A6B2 , B=>S8 , CIN=>C14 , S=>S15 , COUT=>C15);
320  S22_1 : Fulladder port map (A=>A5B3 , B=>S15 , CIN=>C21 , S=>S22 , COUT=>C22);
321  S29_1 : Fulladder port map (A=>A4B4 , B=>S22 , CIN=>C28 , S=>S29 , COUT=>C29);
322  S36_1 : Fulladder port map (A=>A3B5 , B=>S29 , CIN=>C35 , S=>S36 , COUT=>C36);
323  S43_1 : Fulladder port map (A=>A2B6 , B=>S36 , CIN=>C42 , S=>S43 , COUT=>C43);
324  P8 : Fulladder port map (A=>A1B7 , B=>S43 , CIN=>C49 , S=>P(8) , COUT=>C50);

```

Figure 59 Braun code

```

314 S35_1 : Fulladder port map (A=>A2B5 , B=>S28 , CIN=>C34 , S=>S35 , COUT=>C35);
315 S42_1 : Fulladder port map (A=>A1B6 , B=>S35 , CIN=>C41 , S=>S42 , COUT=>C42);
316 P7 : Halfadder port map (A=>A0B7 , B=>S42 , S=>P(7) , COUT=>C49);
317
318 S8_1 : Halfadder port map (A=>A7B1 , B=>C7 , S=>S8 , COUT=>C8);--P8
319 S15_1 : Fulladder port map (A=>A6B2 , B=>S8 , CIN=>C14 , S=>S15 , COUT=>C15);
320 S22_1 : Fulladder port map (A=>A5B3 , B=>S15 , CIN=>C21 , S=>S22 , COUT=>C22);
321 S29_1 : Fulladder port map (A=>A4B4 , B=>S22 , CIN=>C28 , S=>S29 , COUT=>C29);
322 S36_1 : Fulladder port map (A=>A3B5 , B=>S29 , CIN=>C35 , S=>S36 , COUT=>C36);
323 S43_1 : Fulladder port map (A=>A2B6 , B=>S36 , CIN=>C42 , S=>S43 , COUT=>C43);
324 P8 : Fulladder port map (A=>A1B7 , B=>S43 , CIN=>C49 , S=>P(8) , COUT=>C50);
325
326 S16_1 : Fulladder port map (A=>A7B2 , B=>C8 , CIN=>C15 , S=>S16 , COUT=>C16);--P9
327 S23_1 : Fulladder port map (A=>A6B3 , B=>S16 , CIN=>C22 , S=>S23 , COUT=>C23);
328 S30_1 : Fulladder port map (A=>A5B4 , B=>S23 , CIN=>C29 , S=>S30 , COUT=>C30);
329 S37_1 : Fulladder port map (A=>A4B5 , B=>S30 , CIN=>C36 , S=>S37 , COUT=>C37);
330 S44_1 : Fulladder port map (A=>A3B6 , B=>S37 , CIN=>C43 , S=>S44 , COUT=>C44);
331 P9 : Fulladder port map (A=>A2B7 , B=>S44 , CIN=>C50 , S=>P(9) , COUT=>C51);
332
333 S24_1 : Fulladder port map (A=>A7B3 , B=>C16 , CIN=>C23 , S=>S24 , COUT=>C24);--P10
334 S31_1 : Fulladder port map (A=>A6B4 , B=>S24 , CIN=>C30 , S=>S31 , COUT=>C31);
335 S38_1 : Fulladder port map (A=>A5B5 , B=>S31 , CIN=>C37 , S=>S38 , COUT=>C38);
336 S45_1 : Fulladder port map (A=>A4B6 , B=>S38 , CIN=>C44 , S=>S45 , COUT=>C45);
337 P10 : Fulladder port map (A=>A3B7 , B=>S45 , CIN=>C51 , S=>P(10) , COUT=>C52);
338
339 S32_1 : Fulladder port map (A=>A7B4 , B=>C24 , CIN=>C31 , S=>S32 , COUT=>C32);--P11
340 S39_1 : Fulladder port map (A=>A6B5 , B=>S32 , CIN=>C38 , S=>S39 , COUT=>C39);
341 S46_1 : Fulladder port map (A=>A5B6 , B=>S39 , CIN=>C45 , S=>S46 , COUT=>C46);
342 P11 : Fulladder port map (A=>A4B7 , B=>S46 , CIN=>C52 , S=>P(11) , COUT=>C53);
343
344 S40_1 : Fulladder port map (A=>A7B5 , B=>C32 , CIN=>C39 , S=>S40 , COUT=>C40);--P12
345 S47_1 : Fulladder port map (A=>A6B6 , B=>S40 , CIN=>C46 , S=>S47 , COUT=>C47);
346 P12 : Fulladder port map (A=>A5B7 , B=>S47 , CIN=>C53 , S=>P(12) , COUT=>C54);
347
348 S48_1 : Fulladder port map (A=>A7B6 , B=>C40 , CIN=>C47 , S=>S48 , COUT=>C48);--P13
349 P13 : Fulladder port map (A=>A6B7 , B=>S48 , CIN=>C54 , S=>P(13) , COUT=>C55);
350
351 P14 : Fulladder port map (A=>A7B7 , B=>C48 , CIN=>C55 , S=>P(14) , COUT=>C56);--P14
352
353 P15 : P(15) <= C56;--P15
354
355 end Behavioral;
356
357

```

Figure 60 Braun code

Wallace Tree Code

```
1  -- Company:
2  -- Engineer: Luis Lloyd
3  -- Create Date: 14.04.25
4  -- Design Name: Wallace Tree
5  -- Module Name: Wallace_Tree
6  -- Project Name: Electronics II
7
8  library IEEE;
9  use IEEE.STD_LOGIC_1164.ALL;
10
11 entity Wallace_Tree is
12     Port (
13         A : in STD_LOGIC_VECTOR(7 downto 0);
14         B : in STD_LOGIC_VECTOR(7 downto 0);
15         P : out STD_LOGIC_VECTOR(15 downto 0)
16     );
17 end Wallace_Tree;
18
19 architecture Behavioral of Wallace_Tree is
20
21     component Full_Addder is
22         Port (
23             A : in STD_LOGIC;
24             B : in STD_LOGIC;
25             CIN : in STD_LOGIC;
26             S : out STD_LOGIC;
27             COUT : out STD_LOGIC
28         );
29     end component;
30
31     component Half_Addder is
32         Port (
33             A : in STD_LOGIC;
34             B : in STD_LOGIC;
35             S : out STD_LOGIC;
```

Figure 61 Wallace Tree Code

```

30  segment Self_Address
31      Part 6
32      A = 10 * 8192 * 10000
33      B = 10 * 8192 * 10000
34      C = 10 * 8192 * 10000
35      COUNT = 0
36      COUNT = 0
37      end segment
38
39  @begin
40  @begin
41  @begin
42  @begin
43  @begin
44  @begin
45  @begin
46  @begin
47  @begin
48  @begin
49  @begin
50  @begin
51  @begin
52  @begin
53  @begin
54  @begin
55  @begin
56  @begin
57  @begin
58  @begin
59  @begin
60  @begin
61  @begin
62  @begin
63  @begin
64  @begin
65  @begin
66  @begin
67  @begin
68  @begin
69  @begin
70  @begin
71  @begin
72  @begin
73  @begin
74  @begin
75  @begin
76  @begin
77  @begin
78  @begin
79  @begin
80  @begin
81  @begin
82  @begin
83  @begin
84  @begin
85  @begin
86  @begin
87  @begin
88  @begin
89  @begin
90  @begin
91  @begin
92  @begin
93  @begin
94  @begin
95  @begin
96  @begin
97  @begin
98  @begin
99  @begin
100  @begin
101  @begin
102  @begin
103  @begin
104  @begin
105  @begin
106  @begin
107  @begin
108  @begin
109  @begin
110  @begin
111  @begin
112  @begin
113  @begin
114  @begin
115  @begin
116  @begin
117  @begin
118  @begin
119  @begin
120  @begin
121  @begin
122  @begin
123  @begin
124  @begin
125  @begin
126  @begin
127  @begin
128  @begin
129  @begin
130  @begin
131  @begin
132  @begin
133  @begin
134  @begin
135  @begin
136  @begin
137  @begin
138  @begin
139  @begin
140  @begin
141  @begin
142  @begin
143  @begin
144  @begin
145  @begin
146  @begin
147  @begin
148  @begin
149  @begin
150  @begin
151  @begin
152  @begin
153  @begin
154  @begin
155  @begin
156  @begin
157  @begin
158  @begin
159  @begin
160  @begin
161  @begin
162  @begin
163  @begin
164  @begin
165  @begin
166  @begin
167  @begin
168  @begin
169  @begin
170  @begin
171  @begin
172  @begin
173  @begin
174  @begin
175  @begin
176  @begin
177  @begin
178  @begin
179  @begin
180  @begin
181  @begin
182  @begin
183  @begin
184  @begin
185  @begin
186  @begin
187  @begin
188  @begin
189  @begin
190  @begin
191  @begin
192  @begin
193  @begin
194  @begin
195  @begin
196  @begin
197  @begin
198  @begin
199  @begin
200  @begin
201  @begin
202  @begin
203  @begin
204  @begin
205  @begin
206  @begin
207  @begin
208  @begin
209  @begin
210  @begin
211  @begin
212  @begin
213  @begin
214  @begin
215  @begin
216  @begin
217  @begin
218  @begin
219  @begin
220  @begin
221  @begin
222  @begin
223  @begin
224  @begin
225  @begin
226  @begin
227  @begin
228  @begin
229  @begin
230  @begin
231  @begin
232  @begin
233  @begin
234  @begin
235  @begin
236  @begin
237  @begin
238  @begin
239  @begin
240  @begin
241  @begin
242  @begin
243  @begin
244  @begin
245  @begin
246  @begin
247  @begin
248  @begin
249  @begin
250  @begin
251  @begin
252  @begin
253  @begin
254  @begin
255  @begin
256  @begin
257  @begin
258  @begin
259  @begin
260  @begin
261  @begin
262  @begin
263  @begin
264  @begin
265  @begin
266  @begin
267  @begin
268  @begin
269  @begin
270  @begin
271  @begin
272  @begin
273  @begin
274  @begin
275  @begin
276  @begin
277  @begin
278  @begin
279  @begin
280  @begin
281  @begin
282  @begin
283  @begin
284  @begin
285  @begin
286  @begin
287  @begin
288  @begin
289  @begin
290  @begin
291  @begin
292  @begin
293  @begin
294  @begin
295  @begin
296  @begin
297  @begin
298  @begin
299  @begin
300  @begin
301  @begin
302  @begin
303  @begin
304  @begin
305  @begin
306  @begin
307  @begin
308  @begin
309  @begin
310  @begin
311  @begin
312  @begin
313  @begin
314  @begin
315  @begin
316  @begin
317  @begin
318  @begin
319  @begin
320  @begin
321  @begin
322  @begin
323  @begin
324  @begin
325  @begin
326  @begin
327  @begin
328  @begin
329  @begin
330  @begin
331  @begin
332  @begin
333  @begin
334  @begin
335  @begin
336  @begin
337  @begin
338  @begin
339  @begin
340  @begin
341  @begin
342  @begin
343  @begin
344  @begin
345  @begin
346  @begin
347  @begin
348  @begin
349  @begin
350  @begin
351  @begin
352  @begin
353  @begin
354  @begin
355  @begin
356  @begin
357  @begin
358  @begin
359  @begin
360  @begin
361  @begin
362  @begin
363  @begin
364  @begin
365  @begin
366  @begin
367  @begin
368  @begin
369  @begin
370  @begin
371  @begin
372  @begin
373  @begin
374  @begin
375  @begin
376  @begin
377  @begin
378  @begin
379  @begin
380  @begin
381  @begin
382  @begin
383  @begin
384  @begin
385  @begin
386  @begin
387  @begin
388  @begin
389  @begin
390  @begin
391  @begin
392  @begin
393  @begin
394  @begin
395  @begin
396  @begin
397  @begin
398  @begin
399  @begin
400  @begin
401  @begin
402  @begin
403  @begin
404  @begin
405  @begin
406  @begin
407  @begin
408  @begin
409  @begin
410  @begin
411  @begin
412  @begin
413  @begin
414  @begin
415  @begin
416  @begin
417  @begin
418  @begin
419  @begin
420  @begin
421  @begin
422  @begin
423  @begin
424  @begin
425  @begin
426  @begin
427  @begin
428  @begin
429  @begin
430  @begin
431  @begin
432  @begin
433  @begin
434  @begin
435  @begin
436  @begin
437  @begin
438  @begin
439  @begin
440  @begin
441  @begin
442  @begin
443  @begin
444  @begin
445  @begin
446  @begin
447  @begin
448  @begin
449  @begin
450  @begin
451  @begin
452  @begin
453  @begin
454  @begin
455  @begin
456  @begin
457  @begin
458  @begin
459  @begin
460  @begin
461  @begin
462  @begin
463  @begin
464  @begin
465  @begin
466  @begin
467  @begin
468  @begin
469  @begin
470  @begin
471  @begin
472  @begin
473  @begin
474  @begin
475  @begin
476  @begin
477  @begin
478  @begin
479  @begin
480  @begin
481  @begin
482  @begin
483  @begin
484  @begin
485  @begin
486  @begin
487  @begin
488  @begin
489  @begin
490  @begin
491  @begin
492  @begin
493  @begin
494  @begin
495  @begin
496  @begin
497  @begin
498  @begin
499  @begin
500  @begin
501  @begin
502  @begin
503  @begin
504  @begin
505  @begin
506  @begin
507  @begin
508  @begin
509  @begin
510  @begin
511  @begin
512  @begin
513  @begin
514  @begin
515  @begin
516  @begin
517  @begin
518  @begin
519  @begin
520  @begin
521  @begin
522  @begin
523  @begin
524  @begin
525  @begin
526  @begin
527  @begin
528  @begin
529  @begin
530  @begin
531  @begin
532  @begin
533  @begin
534  @begin
535  @begin
536  @begin
537  @begin
538  @begin
539  @begin
540  @begin
541  @begin
542  @begin
543  @begin
544  @begin
545  @begin
546  @begin
547  @begin
548  @begin
549  @begin
550  @begin
551  @begin
552  @begin
553  @begin
554  @begin
555  @begin
556  @begin
557  @begin
558  @begin
559  @begin
560  @begin
561  @begin
562  @begin
563  @begin
564  @begin
565  @begin
566  @begin
567  @begin
568  @begin
569  @begin
570  @begin
571  @begin
572  @begin
573  @begin
574  @begin
575  @begin
576  @begin
577  @begin
578  @begin
579  @begin
580  @begin
581  @begin
582  @begin
583  @begin
584  @begin
585  @begin
586  @begin
587  @begin
588  @begin
589  @begin
590  @begin
591  @begin
592  @begin
593  @begin
594  @begin
595  @begin
596  @begin
597  @begin
598  @begin
599  @begin
600  @begin
601  @begin
602  @begin
603  @begin
604  @begin
605  @begin
606  @begin
607  @begin
608  @begin
609  @begin
610  @begin
611  @begin
612  @begin
613  @begin
614  @begin
615  @begin
616  @begin
617
```

Figure 62 Wallace Tree Code

```

70
71 -- STAGE 1
72 P(0) <= A(0) and B(0);
73 ha11 : Half_Adder port map(A1B0, A0B1, P(1), C1);
74 fa12 : Full_Adder port map(A2B0, A1B1, A0B2, S2, C2);
75 fa13 : Full_Adder port map(A3B0, A2B1, A1B2, S3, C3);
76 fa14 : Full_Adder port map(A4B0, A3B1, A2B2, S4, C4);
77 fa15 : Full_Adder port map(A5B0, A4B1, A3B2, S5, C5);
78 fa16 : Full_Adder port map(A6B0, A5B1, A4B2, S6, C6);
79 fa17 : Full_Adder port map(A7B0, A6B1, A5B2, S7, C7);
80 ha18 : Half_Adder port map(A7B1, A6B2, S8, C8);
81
82 -- STAGE 2
83 ha21 : Half_Adder port map(S2, C1, P(2), C9);
84 fa22 : Full_Adder port map(S3, C2, A0B3, S10, C10);
85 fa23 : Full_Adder port map(S4, C3, A1B3, S11, C11);
86 fa24 : Full_Adder port map(S5, C4, A2B3, S12, C12);
87 fa25 : Full_Adder port map(S6, C5, A3B3, S13, C13);
88 fa26 : Full_Adder port map(S7, C6, A4B3, S14, C14);
89 fa27 : Full_Adder port map(S8, C7, A5B3, S15, C15);
90 fa28 : Full_Adder port map(A7B2, C8, A6B3, S16, C16);
91
92 -- STAGE 3
93 ha31 : Half_Adder port map(S10, C9, P(3), C17);
94 fa32 : Full_Adder port map(S11, C10, A0B4, S18, C18);
95 fa33 : Full_Adder port map(S12, C11, A1B4, S19, C19);
96 fa34 : Full_Adder port map(S13, C12, A2B4, S20, C20);
97 fa35 : Full_Adder port map(S14, C13, A3B4, S21, C21);
98 fa36 : Full_Adder port map(S15, C14, A4B4, S22, C22);
99 fa37 : Full_Adder port map(S16, C15, A5B4, S23, C23);
100 fa38 : Full_Adder port map(A7B3, C16, A6B4, S24, C24);
101
102 -- STAGE 4
103 ha41 : Half_Adder port map(S18, C17, P(4), C25);
104 fa42 : Full_Adder port map(S19, C18, A0B5, S26, C26);
105 fa43 : Full_Adder port map(S20, C19, A1B5, S27, C27);
106 fa44 : Full_Adder port map(S21, C20, A2B5, S28, C28);
107 fa45 : Full_Adder port map(S22, C21, A3B5, S29, C29);
108 fa46 : Full_Adder port map(S23, C22, A4B5, S30, C30);
109 fa47 : Full_Adder port map(S24, C23, A5B5, S31, C31);
110 fa48 : Full_Adder port map(A7B4, C24, A6B5, S32, C32);
111
112 -- STAGE 5

```

Figure 63 Wallace Tree Code

```

102      -- STAGE 4
103      ha41 : Half_Adder port map(S18, C17, P(4), C25);
104      fa42 : Full_Adder port map(S19, C18, A0B5, S26, C26);
105      fa43 : Full_Adder port map(S20, C19, A1B5, S27, C27);
106      fa44 : Full_Adder port map(S21, C20, A2B5, S28, C28);
107      fa45 : Full_Adder port map(S22, C21, A3B5, S29, C29);
108      fa46 : Full_Adder port map(S23, C22, A4B5, S30, C30);
109      fa47 : Full_Adder port map(S24, C23, A5B5, S31, C31);
110      fa48 : Full_Adder port map(A7B4, C24, A6B5, S32, C32);
111
112      -- STAGE 5
113      ha51 : Half_Adder port map(S26, C25, P(5), C33);
114      fa52 : Full_Adder port map(S27, C26, A0B6, S34, C34);
115      fa53 : Full_Adder port map(S28, C27, A1B6, S35, C35);
116      fa54 : Full_Adder port map(S29, C28, A2B6, S36, C36);
117      fa55 : Full_Adder port map(S30, C29, A3B6, S37, C37);
118      fa56 : Full_Adder port map(S31, C30, A4B6, S38, C38);
119      fa57 : Full_Adder port map(S32, C31, A5B6, S39, C39);
120      fa58 : Full_Adder port map(A7B5, C32, A6B6, S40, C40);
121
122      -- STAGE 6
123      ha61 : Half_Adder port map(S34, C33, P(6), C41);
124      fa62 : Full_Adder port map(S35, C34, A0B7, S42, C42);
125      fa63 : Full_Adder port map(S36, C35, A1B7, S43, C43);
126      fa64 : Full_Adder port map(S37, C36, A2B7, S44, C44);
127      fa65 : Full_Adder port map(S38, C37, A3B7, S45, C45);
128      fa66 : Full_Adder port map(S39, C38, A4B7, S46, C46);
129      fa67 : Full_Adder port map(S40, C39, A5B7, S47, C47);
130      fa68 : Full_Adder port map(A7B6, C40, A6B7, S48, C48);
131      Warning: Positional port connection in entity/model
132
133      -- STAGE 7
134      ha71 : Half_Adder port map(S42, C41, P(7), C49);
135      fa72 : Full_Adder port map(S43, C42, C49, P(8), C50);
136      fa73 : Full_Adder port map(S44, C43, C50, P(9), C51);
137      fa74 : Full_Adder port map(S45, C44, C51, P(10), C52);
138      fa75 : Full_Adder port map(S46, C45, C52, P(11), C53);
139      fa76 : Full_Adder port map(S47, C46, C53, P(12), C54);
140      fa77 : Full_Adder port map(S48, C47, C54, P(13), C55);
141      fa78 : Full_Adder port map(A7B7, C48, C55, P(14), P(15));
142  end Behavioral;
143

```

Figure 64 Wallace Tree code

Hardware LED Photos

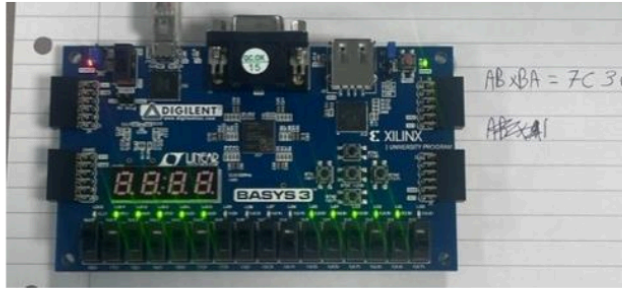


Figure 65 AB x BA Hardware Image

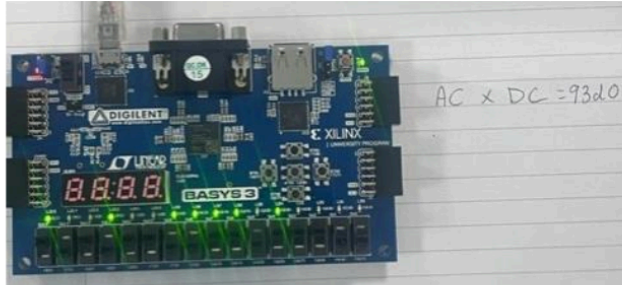


Figure 66 AC x DC Hardware Image

72

/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6



Comment 1 | Engagement with literature(20)

It will be helpful to start with the basic idea of binary multiplication and explain logic circuits like full adder and half adder clearly.



Comment 2 | Engagement with literature(20)

Adding more recent papers and doing a proper review of existing multiplier methods will improve the quality.

PAGE 7



Comment 3 | Knowledge & Understanding(20)

More in-depth explanation of the working process, signal flow, and technical comparisons between the two multipliers would strengthen this section.

PAGE 8



Comment 4 | Engagement with literature(20)

The inclusion of their applications and advantages are appreciable.

PAGE 9



Comment 5 | Practical Application (25)

Give more detail about port mapping.



Comment 6 | Cognitive skills (20)

Brauns multiplier's Partial Product is neat and supports the explanation well.



Comment 7 | Cognitive skills (20)

Wallace multiplier's dot diagram is neat and supports the explanation well.



Comment 8 | Cognitive skills (20)

Architecture diagrams for both Braun and Wallace are shown clearly with labels.



Comment 9 | Knowledge & Understanding(20)

Basic logic circuit explanation and diagrams for binary operations are missing.



Comment 10 | Practical Application (25)

XDC file screenshots are seen, which is appreciable.



Comment 11 | Practical Application (25)

Screenshots like schematics, RTL diagrams, and I/O planning are provided is appreciable.



Comment 12 | Cognitive skills (20)

Critical path calculations for both designs are shown clearly.



Comment 13 | Knowledge & Understanding(20)

Delay
diagrams for both designs are shown clearly.



Comment 14 | Practical Application (25)

Pin
assignments are seen which is appreciable.



Comment 15 | Practical Application (25)

The
report includes timing diagrams and some delay values, which show an effort to
analyse performance.



Comment 16 | Practical Application (25)

Simulation
and hardware results are shown properly, which proves the design works.

PAGE 42

PAGE 43



Comment 17 | Cognitive skills (20)

Comparison of Braun and Wallace multipliers is included for speed, power and area which is excellent.

PAGE 44



Comment 18 | Cognitive skills (20)

The
report presents output performance analysis using parameters such as power, LUT
usage, speed, and area, which indicates that the designs were evaluated
thoroughly.

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49



Comment 19 | Cognitive skills (20)

The
future recommendations are well noted and show a good understanding.

PAGE 50



Comment 20 | Engagement with literature(20)

The
report makes an effort to refer to relevant sources, which shows awareness of
existing work in the field.

PAGE 51



Comment 21 | Transferable skills (15)

The
report is arranged nicely with clear sections.

PAGE 52

PAGE 53

PAGE 54

PAGE 55

GRADING FORM: UNTITLED

LUIS LLOYD

72.10

ENGAGEMENT WITH LITERATURE(20)

Assesses clarity of definition, technical explanation, advantages, disadvantages, applications, and use of relevant references. Also includes critical analysis, research gaps, and proper citation and referencing.



Literature engagement is good, though some references may lack full relevance. Analysis is present but lacks depth in parts. Research gaps are acknowledged but not thoroughly explored. Citations are generally correct, with a few inconsistencies. Source variety is adequate.

12

KNOWLEDGE & UNDERSTANDING(20)

This section evaluates understanding of binary multiplication, multiplier architectures, logic design, and adders. It also includes key concepts like power efficiency, delay path, LUTs, and slack, reflecting application of core engineering principles.



Good basic understanding. Binary multiplication, component usage, and architecture explanations are mostly accurate, but some technical details

13

(e.g., slack or LUTs) may lack depth. Applies general engineering knowledge with limited reference to broader context.

COGNITIVE SKILLS (20)

This section includes analysis of design structures, performance interpretation, and comparison of results using appropriate techniques. Emphasis is placed on understanding technical relationships, interpreting data, and drawing meaningful conclusions with supporting evidence such as calculations, graphs, and recommendation



Shows strong analytical ability. Most design analyses are accurate, with logical interpretation of delay and performance parameters. Graphs or comparisons support conclusions well. Minor details may lack depth, but overall recommendations and insights are valid and technically sound.

15.5

PRACTICAL APPLICATION (25)

This section assesses the practical skills in VHDL implementation, port mapping, FPGA pin assignment, simulation, and hardware proof. It also includes synthesis, applying XDC constraints, and interpreting reports on LUTs, power, delay, and timing, reflecting effective tool usage.



Very good execution of practical tasks. VHDL code is functionally sound; Simulation and hardware outputs are included. FPGA synthesis and report handling are well managed. Constraints are applied with general accuracy.

21.1

TRANSFERABLE SKILLS (15)

This section evaluates the ability to communicate work clearly and effectively. It covers report writing, clarity of diagrams, structured presentation, and appropriate use of screenshots. It also assesses how well findings are explained, the accuracy with which procedures are followed, and how results are summarised. Communication should be

suited to both technical and non-technical audiences, demonstrating professionalism and clarity.



Very good communication. Report is coherent and includes clear diagrams and screenshots. Findings are explained well and structure is mostly consistent. Procedural accuracy and summary are evident. Minor improvements could be made in tailoring language for diverse audiences.

10.5