

Chris James Electronics 2 assignment

by Chris James

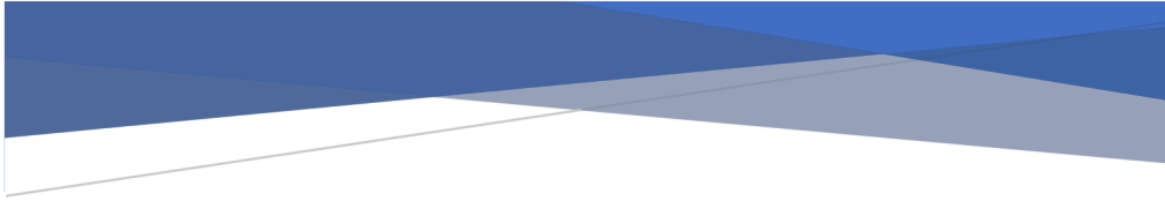
Submission date: 17-Apr-2025 07:01AM (UTC+0100)

Submission ID: 255790317

File name: 50113_Chris_James_Chris_James_Electronics_2_assignment_654553_596671681.docx (5.47M)

Word count: 5784

Character count: 32263



Evaluation and comparison of Brauns and Wallace tree multipliers

By
Christopher James
Electronics 2

Contents

1. Introduction.....	2
2. Research.....	2
2.1 Brauns Multiplier.....	2
2.2 Wallace tree multiplier.....	2
2.3 Conclusion.....	3
2.4. VHDL.....	3
2.5 FPGA.....	4
3. Flowcharts.....	5
3.1 Brauns Multiplier.....	5
3.2 Wallace tree multiplier 8x8.....	6
4. Brauns multiplier code.....	7
5. Wallace tree multiplier code.....	20
6. Testing program – Brauns multiplier.....	29
6.1 AAXAA.....	29
6.2 FFXFF.....	30
6.3 11X11.....	31
6.4 8AX55.....	32
7. Wallace tree multiplier.....	33
7.1 FFXFF.....	33
7.2 AAXAA.....	34
7.3 11x23.....	35
7.4 41x53.....	36
7.5 Conclusion.....	36
8. Brauns and Wallace tree comparisons.....	37
8.1 Brauns.....	37
8.2 Wallace.....	41
8.3 Conclusion.....	45
References.....	46

1. Introduction

This report will compare both Brauns and Wallace multipliers using VHDL (VHSIC Hardware Description Language). To achieve this I will explore the multiplier techniques, advantages of both multipliers and the methodology behind the process of both types of multiplier.

I will show the results of my practical simulations, comparing the results of each multiplier to determine the advantages and disadvantages of both.

2. Research

Multiplication is vital to the functionality of digital systems where signal processing is required. Methods of quick hardware multiplication are important to achieve high efficiency and precision within equipment. Two separate methods of parallel multipliers are Brauns and Wallace tree.

2.1 Brauns Multiplier

Brauns multiplier  (Carry-save multiplier), is a unique tool that can be used mainly for unsigned integer multiplication. Within its coding, partial product is created in parallel and then calculated using carry/save adders. The simplicity of this method makes it a suitable option for large scale integration (VLSI). MY simulations will show however, that it is not the most speed efficient, mainly due to an accumulation of carry delays. Brauns multiplier meets the requirements of many circuits, with it capable of carrying out parallel array multiplication over acceptable speeds. [1]

The structure of the Brauns multiplier consists of an array of AND gates and adders that have a regular and parallel layout. This predictable and methodical structure makes them well suited in hardware implementation. The multiplier begins by using AND gates to generate partial products by performing multiplication [2]. These partial products can then be aligned accordingly and summed using carry-save adders that are arranged systematically. Each row of adders will create a layer of partial products, with the carries moving across and the sums downwards. At the last layer, the final result is created.

2.2 Wallace tree multiplier

To combat the delays, the Wallace tree multiplier was created to increase the speed by reducing the amount of sequential add-on steps required to sum partial products. This is achieved via a complex and efficient hardware that uses layers of AND gates, full and half adders to reduce the number of products as soon as it can, minimising the amount of memory used. The Wallace tree aims to reduce the summing layer by layer, ultimately reducing the amount of overall delay. This makes the Wallace tree multiplier suitable for higher speed applications. The drawback of this method is that it will have a more complicated coding process to create.

The Wallace tree method starts by generating all partial products using AND gates. These partial products are then grouped together into sets and reduced using full and half adders to compress the partial products as soon as possible. Instead of summing rows of product separately, the Wallace tree will organise the bits into columns depending on their bit position. It will then begin the reduction process layer by layer. This layered reduction will greatly reduce the amount of clock cycles required, speeding up the entire process as a whole[3].

By compressing the addition sections into a non-linear method, the Wallace tree multiplier is able to improve performance, making it highly suitable for high-speed arithmetic functions, making it ideal for digital signal processing and real time embedded systems implemented into the FPGA's.

The downside to this method is that the structure is very complex and the need for careful organisation of signals makes it a longer and more difficult process, with greater possibilities for error.

2.3 Conclusion

Both Brauns and Wallace tree multipliers are used for high-speed calculations in digital systems, however they both differ in structure, performance and integration. Brauns multiplier follows a less complex structure, making it easier to implement. This is useful for applications where simplicity in design is important. It uses a fixed array of AND gates and carry-save adders to complete its design. Its simplicity, however, comes at the expense of increased delay time due to its sequential addition process.

The Wallace tree multiplier is designed specifically to increase the processing times of the calculations by carrying out reductions in parallel. Minimising the amount of addition layers using full and half adders will help to improve its efficiency as a whole. This makes it suitable for more high-performance applications. Its complex structure however makes it a more time intensive process to create, with larger possibilities of errors in the programming [4].

The choice between both methods is dependent on the circuit requirements. Braun is useful due to its simplicity and ease of implementation, whereas Wallace is ideal for high speeds and performance however at the cost of increased complexity.

2.4. VHDL

VHDL (VHSIC hardware design language) is a unique and powerful programming language that can be used to describe the structure and behavioural methods of digital circuits and systems, such as techniques used in programming FPGA's (field programmable gate arrays). This allows the user to create detailed models of how the hardware will perform before physically building a system, which can be costly and time consuming [5].

VHDL is widely used when designing systems and equipment that require high accuracy and reliability. These include aerospace, automotive electronics, medical and

telecommunication. IT will allow for precise and accurate timing and logic which is vital for safety sensitive or even performance critical applications.

There are many benefits to using VHDL:

- Allows simulation of specific software to be modelled before wasting time and money on physical hardware
- VHDL can be used to describe the concurrent system.
- Once a computational block has been built (such as my half/full adders) they can be reused in other projects, saving time on the coding.
- VHDL allows you to reduce the number of bugs within your coding early on, with unique typing and coding checks.
- Allows for highly structured and easier to read code.
- Supports modular design – You are able to create reusable components and then break the program down into more manageable blocks.
- Compatibility – VHDL is supported by the majority of major FPGA development tools such as Vivado.
- Testing – VHDL allows users to simulate their work and verify it to ensure it works correctly before carrying out programming on hardware, reducing overall development time and cost.

Although VHDL offers many advantages, it also comes with other disadvantages:

- For beginners, it can be difficult to learn and understand due to its coding and navigation methods.
- Writing simple designs such as AND gates require larger amounts of code.

Unlike other types of coding such as C or python, VHDL does not tell a computer what to do or how to function, instead, it tells a FPGA how it should behave. Using VHDL, you are able to write a description of behaviour using code and then use built in programs to synthesise the code and turn it into combinational logic such as and gates or NOT gates.

In comparison to other programming languages such as C, where you may write a program that will add two numbers, in VHDL, you will describe how the hardware will perform the addition, specifying how the logic will flow through the logic gates to get to the final result. This description is what is then synthesised into logic that will run on the FPGA.

2.5 FPGA

Field programmable gate arrays are flexible, digital devices that will allow the user to create custom hardware formations. Unlike other methods (e.g ASIC), a FPGA can be reprogrammed to perform a wide multitude of tasks. It is one of the most used languages when describing circuits performance. Combining FPGA and VHDL, a user is capable of designing, simulating and creating complicated hardware circuitry to a high level of accuracy and efficiency.

3. Flowcharts

3.1 Brauns Multiplier

The below flowchart shows the Brauns multiplier layout. To create the full circuit, each bit in one row is multiplied with every bit in the other row. (e.g. A0B1). Once complete on the first row, each bit is then shifted and then summed together again, eventually creating a final result of the multiplication.

BRAUNS 8X8 MULTIPLIER

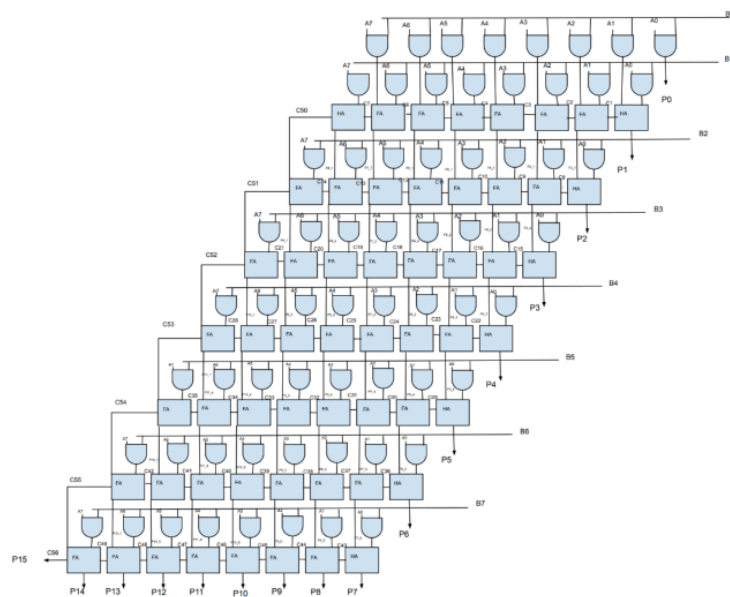


Figure 1 - Brauns multiplier 8x8

3.2 Wallace tree multiplier 8x8

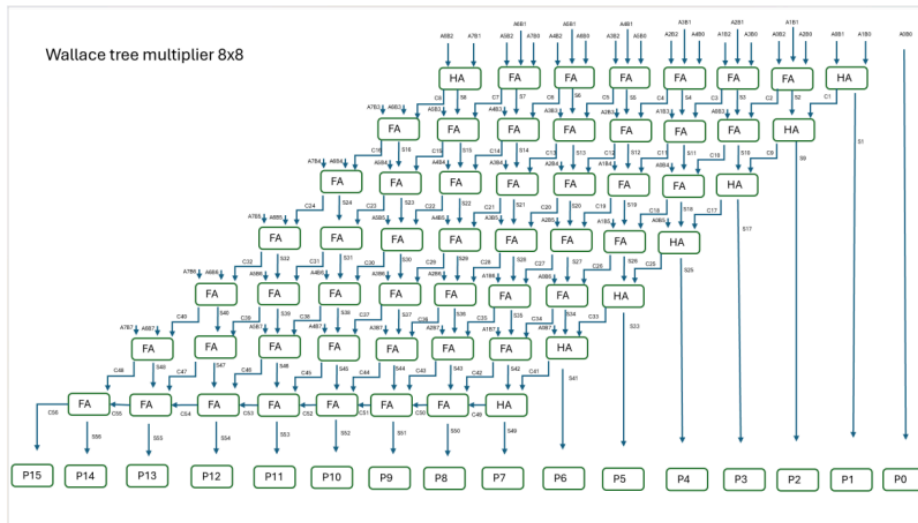


Figure 2 - Wallace tree multiplier 8x8

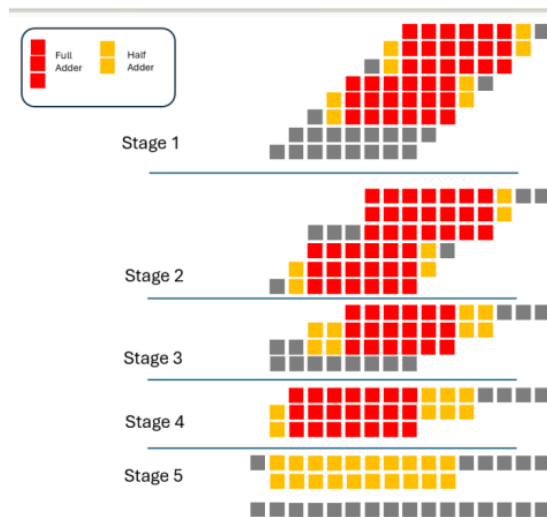


Figure 3 - Wallace dot diagram

Figure 3 demonstrates an 8x8 Wallace multiplier and shows its structure. This shows the placement of the full and half adders to generate partial products and carry out immediate calculations

4. Brauns multiplier code

-- Company:

-- Engineer:

--

-- Create Date: 20.03.2025 12:30:16

-- Design Name:

-- Module Name: Brauns - Behavioural

-- Project Name:

-- Target Devices:

-- Tool Versions:

-- Description:

--

-- Dependencies:

--

-- Revision:

-- Revision 0.01 - File Created

-- Additional Comments:

--

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_logic_arith.all;

use IEEE.std_logic_unsigned.all;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating

-- any Xilinx leaf cells in this code.

--library UNISIM;

--use UNISIM.VComponents.all;

entity Brauns is

Port (A: in std_logic_vector (7 downto 0) ;

B: in std_logic_vector (7 downto 0);

P: out std_logic_vector (15 downto 0));

-- Port ();

end Brauns;

architecture behavioral of Brauns is

signal A1B0 : std_logic;

signal A2B0 : std_logic;

signal A3B0 : std_logic;

signal A4B0 : std_logic;

signal A5B0 : std_logic;

signal A6B0 : std_logic;

signal A7B0 : std_logic;

signal A0B1 : std_logic;

signal A1B1 : std_logic;

signal A2B1 : std_logic;

signal A3B1 : std_logic;

signal A4B1 : std_logic;

signal A5B1 : std_logic;

signal A6B1 : std_logic;

signal A7B1 : std_logic;

signal A0B2 : std_logic;

```
signal A1B2 : std_logic;
signal A2B2 : std_logic;
signal A3B2 : std_logic;
signal A4B2 : std_logic;
signal A5B2 : std_logic;
signal A6B2 : std_logic;
signal A7B2 : std_logic;
signal A0B3 : std_logic;
signal A1B3 : std_logic;
signal A2B3 : std_logic;
signal A3B3 : std_logic;
signal A4B3 : std_logic;
signal A5B3 : std_logic;
signal A6B3 : std_logic;
signal A7B3 : std_logic;
signal A0B4 : std_logic;
signal A1B4 : std_logic;
signal A2B4 : std_logic;
signal A3B4 : std_logic;
signal A4B4 : std_logic;
signal A5B4 : std_logic;
signal A6B4 : std_logic;
signal A7B4 : std_logic;
signal A0B5 : std_logic;
signal A1B5 : std_logic;
signal A2B5 : std_logic;
signal A3B5 : std_logic;
signal A4B5 : std_logic;
signal A5B5 : std_logic;
signal A6B5 : std_logic;
signal A7B5 : std_logic;
```

```
signal A0B6 : std_logic;  
signal A1B6 : std_logic;  
signal A2B6 : std_logic;  
signal A3B6 : std_logic;  
signal A4B6 : std_logic;  
signal A5B6 : std_logic;  
signal A6B6 : std_logic;  
signal A7B6 : std_logic;  
signal A0B7 : std_logic;  
signal A1B7 : std_logic;  
signal A2B7 : std_logic;  
signal A3B7 : std_logic;  
signal A4B7 : std_logic;  
signal A5B7 : std_logic;  
signal A6B7 : std_logic;  
signal A7B7 : std_logic;
```

```
signal C1 : std_logic; --P2  
signal p2_1 : std_logic;
```

```
signal C2 : std_logic; --p3  
signal P3_1 : std_logic;  
signal C8 : std_logic;  
signal P3_2 : std_logic;
```

```
signal C3 : std_logic; --p4  
signal P4_1 : std_logic;  
signal C9 : std_logic;  
signal P4_2 : std_logic;  
signal C15 : std_logic;
```

```
signal P4_3 : std_logic;
```

```
signal C4 : std_logic; --p5
```

```
signal P5_1 : std_logic;
```

```
signal C10 : std_logic;
```

```
signal P5_2 : std_logic;
```

```
signal C16 : std_logic;
```

```
signal P5_3 : std_logic;
```

```
signal C22 : std_logic;
```

```
signal P5_4 : std_logic;
```

```
signal C5 : std_logic; --p6
```

```
signal P6_1 : std_logic;
```

```
signal C11 : std_logic;
```

```
signal P6_2 : std_logic;
```

```
signal C17 : std_logic;
```

```
signal P6_3 : std_logic;
```

```
signal C23 : std_logic;
```

```
signal P6_4 : std_logic;
```

```
signal C29 : std_logic;
```

```
signal P6_5 : std_logic;
```

```
signal C6 : std_logic; --p7
```

```
signal P7_1 : std_logic;
```

```
signal C12 : std_logic;
```

```
signal P7_2 : std_logic;
```

```
signal C18 : std_logic;
```

```
signal P7_3 : std_logic;
```

```
signal C24 : std_logic;
```

```
signal P7_4 : std_logic;
```

```
signal C30 : std_logic;
```

```
signal P7_5 : std_logic;  
signal C36 : std_logic;  
signal P7_6 : std_logic;
```

```
signal C7 : std_logic; --p8  
signal P8_1 : std_logic;  
signal C13 : std_logic;  
signal P8_2 : std_logic;  
signal C19 : std_logic;  
signal P8_3 : std_logic;  
signal C25 : std_logic;  
signal P8_4 : std_logic;  
signal C31 : std_logic;  
signal P8_5 : std_logic;  
signal C37 : std_logic;  
signal P8_6 : std_logic;  
signal C43 : std_logic;
```

```
signal C50 : std_logic; --p9  
signal P9_1 : std_logic;  
signal C14 : std_logic;  
signal P9_2 : std_logic;  
signal C20 : std_logic;  
--signal P9_3 : std_logic;  
signal C26 : std_logic;  
signal P9_4 : std_logic;  
signal C32 : std_logic;  
signal P9_5 : std_logic;  
signal C38 : std_logic;  
signal P9_6 : std_logic;  
signal C44 : std_logic;
```

signal C51 : std_logic; --p10

signal C21 : std_logic;

signal P10_1 : std_logic;

signal C27 : std_logic;

signal P10_4 : std_logic;

signal C33 : std_logic;

signal P10_5 : std_logic;

signal C39 : std_logic;

signal P10_6 : std_logic;

signal C45 : std_logic;

signal C52 : std_logic; --p11

signal C28 : std_logic;

signal P11_4 : std_logic;

signal C34 : std_logic;

signal P11_5 : std_logic;

signal C40 : std_logic;

signal P11_6 : std_logic;

signal C46 : std_logic;

signal C53 : std_logic; --p12

signal C35 : std_logic;

signal P12_1 : std_logic;

signal C41 : std_logic;

signal P12_6 : std_logic;

signal C47 : std_logic;

signal C54 : std_logic; --p13

signal C42 : std_logic;

signal P13_1 : std_logic;

```
signal C48 : std_logic;
```

```
signal C55 : std_logic; --p14
```

```
signal C49 : std_logic;
```

```
signal C56 : std_logic; --P15
```

```
component Full_adder
```

```
Port ( A : in std_logic;
```

```
B : in std_logic;
```

```
Cin : in std_logic;
```

```
S : out std_logic;
```

```
Cout : out std_logic);
```

```
end component;
```

```
component half_adder
```

```
port (A : in std_logic;
```

```
B : in std_logic;
```

```
S : out std_logic;
```

```
Cout : out std_logic);
```

```
end component;
```

```
begin
```

```
A0B1 <= A(0) and B(1);
```

```
A0B2 <= A(0) and B(2);
```

```
A0B3 <= A(0) and B(3);
```

```
A0B4 <= A(0) and B(4);
```

```
A0B5 <= A(0) and B(5);
```

A0B6 <= A(0) and B(6);
A0B7 <= A(0) and B(7);
A1B0 <= A(1) and B(0);
A1B1 <= A(1) and B(1);
A1B2 <= A(1) and B(2);
A1B3 <= A(1) and B(3);
A1B4 <= A(1) and B(4);
A1B5 <= A(1) and B(5);
A1B6 <= A(1) and B(6);
A1B7 <= A(1) and B(7);
A2B0 <= A(2) and B(0);
A2B1 <= A(2) and B(1);
A2B2 <= A(2) and B(2);
A2B3 <= A(2) and B(3);
A2B4 <= A(2) and B(4);
A2B5 <= A(2) and B(5);
A2B6 <= A(2) and B(6);
A2B7 <= A(2) and B(7);
A3B0 <= A(3) and B(0);
A3B1 <= A(3) and B(1);
A3B2 <= A(3) and B(2);
A3B3 <= A(3) and B(3);
A3B4 <= A(3) and B(4);
A3B5 <= A(3) and B(5);
A3B6 <= A(3) and B(6);
A3B7 <= A(3) and B(7);
A4B0 <= A(4) and B(0);
A4B1 <= A(4) and B(1);
A4B2 <= A(4) and B(2);
A4B3 <= A(4) and B(3);
A4B4 <= A(4) and B(4);

A4B5 <= A(4) and B(5);

A4B6 <= A(4) and B(6);

A4B7 <= A(4) and B(7);

A5B0 <= A(5) and B(0);

A5B1 <= A(5) and B(1);

A5B2 <= A(5) and B(2);

A5B3 <= A(5) and B(3);

A5B4 <= A(5) and B(4);

A5B5 <= A(5) and B(5);

A5B6 <= A(5) and B(6);

A5B7 <= A(5) and B(7);

A6B0 <= A(6) and B(0);

A6B1 <= A(6) and B(1);

A6B2 <= A(6) and B(2);

A6B3 <= A(6) and B(3);

A6B4 <= A(6) and B(4);

A6B5 <= A(6) and B(5);

A6B6 <= A(6) and B(6);

A6B7 <= A(6) and B(7);

A7B0 <= A(7) and B(0);

A7B1 <= A(7) and B(1);

A7B2 <= A(7) and B(2);

A7B3 <= A(7) and B(3);

A7B4 <= A(7) and B(4);

A7B5 <= A(7) and B(5);

A7B6 <= A(7) and B(6);

A7B7 <= A(7) and B(7);

P0 : P(0) <= A(0) and B(0);

P1_0 : Half_adder port map (A => A1B0, B=>A0B1, S=> P(1), Cout=>C1);

P21 : Full_adder port map (A=>A2B0, B=>A1B1, Cin=>C1, S=>P2_1, Cout=>C2);

P2_0 : Half_adder port map (A=>A0B2, B=>P2_1, S=>P(2), Cout=>C8);

P31 : full_adder port map (A=> A3B0, B=> A2B1, Cin=>C2, S=>P3_1, Cout=>C3);

P32 : full_adder port map (A=>A1B2, B=>P3_1, Cin=>C8, S=>P3_2, Cout=>C9);

P3_0 : half_adder port map (A=>A0B3, B=>P3_2, S=>P(3), Cout=>C15);

P41 : full_adder port map (A=> A4B0, B=> A3B1, Cin=>C3, S=>P4_1, Cout=>C4);

P42 : full_adder port map (A=>A2B2, B=>P4_1, Cin=>C9, S=>P4_2, Cout=>C10);

P43 : full_adder port map (A=>A1B3, B=>P4_2, Cin=>C15, S=>P4_3, Cout=>C16);

P4_0 : half_adder port map (A=>A0B4, B=>P4_3, S=>P(4), Cout=>C22);

P51 : full_adder port map (A=> A5B0, B=> A4B1, Cin=>C4, S=>P5_1, Cout=>C5);

P52 : full_adder port map (A=>A3B2, B=>P5_1, Cin=>C10, S=>P5_2, Cout=>C11);

P53 : full_adder port map (A=>A2B3, B=>P5_2, Cin=>C16, S=>P5_3, Cout=>C17);

P54 : full_adder port map (A=>A1B4, B=>P5_3, Cin=>C22, S=>P5_4, Cout=>C23);

P5_0 : half_adder port map (A=>A0B5, B=>P5_4, S=>P(5), Cout=>C29);

P61 : full_adder port map (A=> A6B0, B=> A5B1, Cin=>C5, S=>P6_1, Cout=>C6);

P62 : full_adder port map (A=>A4B2, B=>P6_1, Cin=>C11, S=>P6_2, Cout=>C12);

P63 : full_adder port map (A=>A3B3, B=>P6_2, Cin=>C17, S=>P6_3, Cout=>C18);

P64 : full_adder port map (A=>A2B4, B=>P6_3, Cin=>C23, S=>P6_4, Cout=>C24);

P65 : full_adder port map (A=>A1B5, B=>P6_4, Cin=>C29, S=>P6_5, Cout=>C30);

P6_0 : half_adder port map (A=>A0B6, B=>P6_5, S=>P(6), Cout=>C36);

P71 : full_adder port map (A=> A7B0, B=> A6B1, Cin=>C6, S=>P7_1, Cout=>C7);

P72 : full_adder port map (A=>A5B2, B=>P7_1, Cin=>C11, S=>P7_2, Cout=>C13);

P73 : full_adder port map (A=>A4B3, B=>P7_2, Cin=>C17, S=>P7_3, Cout=>C19);

P74 : full_adder port map (A=>A3B4, B=>P7_3 , Cin=>C23, S=>P7_4, Cout=>C25);
P75 : full_adder port map (A=>A2B5, B=>P7_4 , Cin=>C29, S=>P7_5, Cout=>C31);
P76 : full_adder port map (A=>A1B5, B=>P7_5 , Cin=>C29, S=>P7_6, Cout=>C37);
P7_0 : half_adder port map (A=>A0B6, B=>P7_6, S=>P(7), Cout=>C43);

P81 : half_adder port map (A=>A7B1, B=>C7, S=>P8_1, Cout=>C50);
P82 : full_adder port map (A=>A6B2, B=>P8_1 , Cin=>C13, S=>P8_2, Cout=>C14);
P83 : full_adder port map (A=>A5B3, B=>P8_2 , Cin=>C19, S=>P8_3, Cout=>C20);
P84 : full_adder port map (A=>A4B4, B=>P8_3 , Cin=>C25, S=>P8_4, Cout=>C26);
P85 : full_adder port map (A=>A3B5, B=>P8_4 , Cin=>C31, S=>P8_5, Cout=>C32);
P86 : full_adder port map (A=>A2B6, B=>P8_5 , Cin=>C37, S=>P8_6, Cout=>C38);
P8_0 : full_adder port map (A=>A1B7, B=>P8_6 , Cin=>C43, S=>P(8), Cout=>C44);

P91 : full_adder port map (A=>A7B2, B=>C50 , Cin=>C14, S=>P9_1, Cout=>C51);
P92 : full_adder port map (A=>A6B3, B=>P9_1 , Cin=>C20, S=>P9_2, Cout=>C21);
P93 : full_adder port map (A=>A5B4, B=>P9_2 , Cin=>C26, S=>P9_4, Cout=>C27);
P94 : full_adder port map (A=>A4B5, B=>P9_4 , Cin=>C32, S=>P9_5, Cout=>C33);
P95 : full_adder port map (A=>A3B6, B=>P9_5 , Cin=>C38, S=>P9_6, Cout=>C39);
P9_0 : full_adder port map (A=>A2B7, B=>P9_6 , Cin=>C44, S=>P(9), Cout=>C45);

P101 : full_adder port map (A=>A7B3, B=>C51 , Cin=>C21, S=>P10_1, Cout=>C52);
P102 : full_adder port map (A=>A6B4, B=>P10_1 , Cin=>C27, S=>P10_4, Cout=>C28);
P103 : full_adder port map (A=>A5B5, B=>P10_4 , Cin=>C33, S=>P10_5, Cout=>C34);
P104 : full_adder port map (A=>A4B6, B=>P10_5 , Cin=>C39, S=>P10_6, Cout=>C40);
P10_0 : full_adder port map (A=>A3B7, B=>P10_6 , Cin=>C45, S=>P(10), Cout=>C46);

P111 : full_adder port map (A=>A7B4, B=>C52 , Cin=>C28, S=>P11_4, Cout=>C53);
P112 : full_adder port map (A=>A6B5, B=>P11_4 , Cin=>C34, S=>P11_5, Cout=>C35);
P113 : full_adder port map (A=>A5B6, B=>P11_5 , Cin=>C40, S=>P11_6, Cout=>C41);
P11_0 : full_adder port map (A=>A4B7, B=>P11_6 , Cin=>C46, S=>P(11), Cout=>C47);

```
P121 : full_adder port map (A=>A7B5, B=>C53 , Cin=>C35, S=>P12_1, Cout=>C54);
P122 : full_adder port map (A=>A6B6, B=>P12_1 , Cin=>C41, S=>P12_6, Cout=>C42);
P12_0 : full_adder port map (A=>A5B7, B=>P12_6 , Cin=>C47, S=>P(12), Cout=>C48);

P131 : full_adder port map (A=>A7B6, B=>C54 , Cin=>C42, S=>P13_1, Cout=>C55);
P13_0 : full_adder port map (A=>A6B7, B=>P13_1 , Cin=>C48, S=>P(13), Cout=>C49);

P14_0 : full_adder port map (A=>A7B7, B=>C55 , Cin=>C49, S=>P(14), Cout=>C56);

P15_0 : P(15) <= C56;
```

```
end Behavioral;
```

5. Wallace tree multiplier code

```
-- Company:
-- Engineer:
--
-- Create Date: 27.03.2025 16:50:10
-- Design Name:
-- Module Name: Wallace - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
```

```
library IEEE;
```

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

```
entity wallace is
```

```
    Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
```

```
          B : in  STD_LOGIC_VECTOR (7 downto 0);
```

```
          P : out STD_LOGIC_VECTOR (15 downto 0));
```

```
end wallace;
```

architecture Behavioral of wallace is

component Full_adder is

```
Port ( A : in STD_LOGIC;
```

```
      B : in STD_LOGIC;
```

```
      Cin : in STD_LOGIC;
```

```
      S : out STD_LOGIC;
```

```
      Cout : out STD_LOGIC);
```

```
end component;
```

component Half_adder is

```
Port ( A : in STD_LOGIC;
```

```
      B : in STD_LOGIC;
```

```
      S : out STD_LOGIC;
```

```
      Cout : out STD_LOGIC);
```

```
end component;
```

signal

```
S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15,S16,S17,S18,S19,S20,S21,S22,S23,S24,S25,S26,S27,  
S28,S29,S30,S31,S32,S33,S34,S35,S36,S37,S38,S39,S40,S41,S42,S43,S44,S45,S46,S47,S48: std_logic;
```

signal

```
C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16,C17,C18,C19,C20,C21,C22,C23,C24,C25,C  
26,C27,C28,C29,C30,C31,C32,C33,C34,C35,C36,C37,C38,C39,C40,C41,C42,C43,C44,C45,C46,C47,C48,  
C49,C50,C51,C52,C53,C54,C55: std_logic;
```

```
signal A1B0 : std_logic;
```

```
signal A2B0 : std_logic;
```

```
signal A3B0 : std_logic;
```

```
signal A4B0 : std_logic;
```

```
signal A5B0 : std_logic;
```

signal A6B0 : std_logic;

signal A7B0 : std_logic;

signal A0B1 : std_logic;

signal A1B1 : std_logic;

signal A2B1 : std_logic;

signal A3B1 : std_logic;

signal A4B1 : std_logic;

signal A5B1 : std_logic;

signal A6B1 : std_logic;

signal A7B1 : std_logic;

signal A0B2 : std_logic;

signal A1B2 : std_logic;

signal A2B2 : std_logic;

signal A3B2 : std_logic;

signal A4B2 : std_logic;

signal A5B2 : std_logic;

signal A6B2 : std_logic;

signal A7B2 : std_logic;

signal A0B3 : std_logic;

signal A1B3 : std_logic;

signal A2B3 : std_logic;

signal A3B3 : std_logic;

signal A4B3 : std_logic;

signal A5B3 : std_logic;

signal A6B3 : std_logic;

signal A7B3 : std_logic;

signal A0B4 : std_logic;

```
signal A1B4 : std_logic;  
signal A2B4 : std_logic;  
signal A3B4 : std_logic;  
signal A4B4 : std_logic;  
signal A5B4 : std_logic;  
signal A6B4 : std_logic;  
signal A7B4 : std_logic;
```

```
signal A0B5 : std_logic;  
signal A1B5 : std_logic;  
signal A2B5 : std_logic;  
signal A3B5 : std_logic;  
signal A4B5 : std_logic;  
signal A5B5 : std_logic;  
signal A6B5 : std_logic;  
signal A7B5 : std_logic;
```

```
signal A0B6 : std_logic;  
signal A1B6 : std_logic;  
signal A2B6 : std_logic;  
signal A3B6 : std_logic;  
signal A4B6 : std_logic;  
signal A5B6 : std_logic;  
signal A6B6 : std_logic;  
signal A7B6 : std_logic;
```

```
signal A0B7 : std_logic;  
signal A1B7 : std_logic;  
signal A2B7 : std_logic;  
signal A3B7 : std_logic;  
signal A4B7 : std_logic;
```

```
signal A5B7 : std_logic;
```

```
signal A6B7 : std_logic;
```

```
signal A7B7 : std_logic;
```

```
begin
```

```
A1B0 <= A(1) and B(0);
```

```
A2B0 <= A(2) and B(0);
```

```
A3B0 <= A(3) and B(0);
```

```
A4B0 <= A(4) and B(0);
```

```
A5B0 <= A(5) and B(0);
```

```
A6B0 <= A(6) and B(0);
```

```
A7B0 <= A(7) and B(0);
```

```
A0B1 <= A(0) and B(1);
```

```
A1B1 <= A(1) and B(1);
```

```
A2B1 <= A(2) and B(1);
```

```
A3B1 <= A(3) and B(1);
```

```
A4B1 <= A(4) and B(1);
```

```
A5B1 <= A(5) and B(1);
```

```
A6B1 <= A(6) and B(1);
```

```
A7B1 <= A(7) and B(1);
```

```
A0B2 <= A(0) and B(2);
```

```
A1B2 <= A(1) and B(2);
```

```
A2B2 <= A(2) and B(2);
```

```
A3B2 <= A(3) and B(2);
```

```
A4B2 <= A(4) and B(2);
```

A5B2 <= A(5) and B(2);

A6B2 <= A(6) and B(2);

A7B2 <= A(7) and B(2);

A0B3 <= A(0) and B(3);

A1B3 <= A(1) and B(3);

A2B3 <= A(2) and B(3);

A3B3 <= A(3) and B(3);

A4B3 <= A(4) and B(3);

A5B3 <= A(5) and B(3);

A6B3 <= A(6) and B(3);

A7B3 <= A(7) and B(3);

A0B4 <= A(0) and B(4);

A1B4 <= A(1) and B(4);

A2B4 <= A(2) and B(4);

A3B4 <= A(3) and B(4);

A4B4 <= A(4) and B(4);

A5B4 <= A(5) and B(4);

A6B4 <= A(6) and B(4);

A7B4 <= A(7) and B(4);

A0B5 <= A(0) and B(5);

A1B5 <= A(1) and B(5);

A2B5 <= A(2) and B(5);

A3B5 <= A(3) and B(5);

A4B5 <= A(4) and B(5);

A5B5 <= A(5) and B(5);

A6B5 <= A(6) and B(5);

A7B5 <= A(7) and B(5);

```
A0B6 <= A(0) and B(6);
A1B6 <= A(1) and B(6);
A2B6 <= A(2) and B(6);
A3B6 <= A(3) and B(6);
A4B6 <= A(4) and B(6);
A5B6 <= A(5) and B(6);
A6B6 <= A(6) and B(6);
A7B6 <= A(7) and B(6);
```

```
A0B7 <= A(0) and B(7);
A1B7 <= A(1) and B(7);
A2B7 <= A(2) and B(7);
A3B7 <= A(3) and B(7);
A4B7 <= A(4) and B(7);
A5B7 <= A(5) and B(7);
A6B7 <= A(6) and B(7);
A7B7 <= A(7) and B(7);
```

--first stage

```
P0      : P(0) <= A(0) and B(0) ;
```

```
ha11 : Half_adder port map(A=>A1B0,B=>A0B1,S=>P(1),Cout=>C1);
fa12 : Full_adder port map(A=>A2B0,B=>A1B1,Cin=>A0B2,S=>S2,Cout=>C2);
fa13 : Full_adder port map(A=>A3B0,B=>A2B1,Cin=>A1B2,S=>S3,Cout=>C3);
fa14 : Full_adder port map(A=>A4B0,B=>A3B1,Cin=>A2B2,S=>S4,Cout=>C4);
fa15 : Full_adder port map(A=>A5B0,B=>A4B1,Cin=>A3B2,S=>S5,Cout=>C5);
fa16 : Full_adder port map(A=>A6B0,B=>A5B1,Cin=>A4B2,S=>S6,Cout=>C6);
fa17 : Full_adder port map(A=>A7B0,B=>A6B1,Cin=>A5B2,S=>S7,Cout=>C7);
ha18 : Half_adder port map(A=>A7B1,B=>A6B2,S=>S8,Cout=>C8);
```

-- Second stage

```
ha21 : Half_adder port map(A=>S2,B=>C1,S=>P(2),Cout=>C9);
fa22 : Full_adder port map(A=>S3,B=>C2,Cin=>A0B3,S=>S10,Cout=>C10);
fa23 : Full_adder port map(A=>S4,B=>C3,Cin=>A1B3,S=>S11,Cout=>C11);
fa24 : Full_adder port map(A=>S5,B=>C4,Cin=>A2B3,S=>S12,Cout=>C12);
fa25 : Full_adder port map(A=>S6,B=>C5,Cin=>A3B3,S=>S13,Cout=>C13);
fa26 : Full_adder port map(A=>S7,B=>C6,Cin=>A4B3,S=>S14,Cout=>C14);
fa27 : Full_adder port map(A=>S8,B=>C7,Cin=>A5B3,S=>S15,Cout=>C15);
fa28 : Full_adder port map(A=>A7B2,B=>C8,Cin=>A6B3,S=>S16,Cout=>C16);
```

-- Third stage

```
ha31 : Half_adder port map(A=>S10,B=>C9,S=>P(3),Cout=>C17);
fa32 : Full_adder port map(A=>S11,B=>C10,Cin=>A0B4,S=>S18,Cout=>C18);
fa33 : Full_adder port map(A=>S12,B=>C11,Cin=>A1B4,S=>S19,Cout=>C19);
fa34 : Full_adder port map(A=>S13,B=>C12,Cin=>A2B4,S=>S20,Cout=>C20);
fa35 : Full_adder port map(A=>S14,B=>C13,Cin=>A3B4,S=>S21,Cout=>C21);
fa36 : Full_adder port map(A=>S15,B=>C14,Cin=>A4B4,S=>S22,Cout=>C22);
fa37 : Full_adder port map(A=>S16,B=>C15,Cin=>A5B4,S=>S23,Cout=>C23);
fa38 : Full_adder port map(A=>A7B3,B=>C16,Cin=>A6B4,S=>S24,Cout=>C24);
```

-- fourth stage

```
ha41 : Half_adder port map(A=>S18,B=>C17,S=>P(4),Cout=>C25);
fa42 : Full_adder port map(A=>S19,B=>C18,Cin=>A0B5,S=>S26,Cout=>C26);
fa43 : Full_adder port map(A=>S20,B=>C19,Cin=>A1B5,S=>S27,Cout=>C27);
fa44 : Full_adder port map(A=>S21,B=>C20,Cin=>A2B5,S=>S28,Cout=>C28);
fa45 : Full_adder port map(A=>S22,B=>C21,Cin=>A3B5,S=>S29,Cout=>C29);
fa46 : Full_adder port map(A=>S23,B=>C22,Cin=>A4B5,S=>S30,Cout=>C30);
fa47 : Full_adder port map(A=>S24,B=>C23,Cin=>A5B5,S=>S31,Cout=>C31);
fa48 : Full_adder port map(A=>A7B4,B=>C24,Cin=>A6B5,S=>S32,Cout=>C32);
```

-- Fifth Stage

```

ha51 : Half_adder port map(A=>S26,B=>C25,S=>P(5),Cout=>C33);
fa52 : Full_adder port map(A=>S27,B=>C26,Cin=>A0B6,S=>S34,Cout=>C34);
fa53 : Full_adder port map(A=>S28,B=>C27,Cin=>A1B6,S=>S35,Cout=>C35);
fa54 : Full_adder port map(A=>S29,B=>C28,Cin=>A2B6,S=>S36,Cout=>C36);
fa55 : Full_adder port map(A=>S30,B=>C29,Cin=>A3B6,S=>S37,Cout=>C37);
fa56 : Full_adder port map(A=>S31,B=>C30,Cin=>A4B6,S=>S38,Cout=>C38);
fa57 : Full_adder port map(A=>S32,B=>C31,Cin=>A5B6,S=>S39,Cout=>C39);
fa58 : Full_adder port map(A=>A7B5,B=>C32,Cin=>A6B6,S=>S40,Cout=>C40);

```

-- Sixth Stage

```

ha61 : Half_adder port map(A=>S34,B=>C33,S=>P(6),Cout=>C41);
fa62 : Full_adder port map(A=>S35,B=>C34,Cin=>A0B7,S=>S42,Cout=>C42);
fa63 : Full_adder port map(A=>S36,B=>C35,Cin=>A1B7,S=>S43,Cout=>C43);
fa64 : Full_adder port map(A=>S37,B=>C36,Cin=>A2B7,S=>S44,Cout=>C44);
fa65 : Full_adder port map(A=>S38,B=>C37,Cin=>A3B7,S=>S45,Cout=>C45);
fa66 : Full_adder port map(A=>S39,B=>C38,Cin=>A4B7,S=>S46,Cout=>C46);
fa67 : Full_adder port map(A=>S40,B=>C39,Cin=>A5B7,S=>S47,Cout=>C47);
fa68 : Full_adder port map(A=>A7B6,B=>C40,Cin=>A6B7,S=>S48,Cout=>C48);

```

--Seventh Stage

```

ha71 : Half_adder port map(A=>S42,B=>C41,S=>P(7),Cout=>C49);
fa72 : Full_adder port map(A=>S43,B=>C42,Cin=>C49,S=>P(8),Cout=>C50);
fa73 : Full_adder port map(A=>S44,B=>C43,Cin=>C50,S=>P(9),Cout=>C51);
fa74 : Full_adder port map(A=>S45,B=>C44,Cin=>C51,S=>P(10),Cout=>C52);
fa75 : Full_adder port map(A=>S46,B=>C45,Cin=>C52,S=>P(11),Cout=>C53);
fa76 : Full_adder port map(A=>S47,B=>C46,Cin=>C53,S=>P(12),Cout=>C54);
fa77 : Full_adder port map(A=>S48,B=>C47,Cin=>C54,S=>P(13),Cout=>C55);
fa78 : Full_adder port map(A=>A7B7,B=>C48,Cin=>C55,S=>P(14),Cout=>P(15));

```

end Behavioral;

6. Testing program – Brauns multiplier

To ensure that the simulation functioned correctly, I carried out multiple equations of different values across the hexadecimal spectrum to ensure that the final result would all be correct. I compared them to an online Hexadecimal calculator [6] to compare values.

To set my simulation, the clock was forced with a leading edge value = 0 and trailing edge value of 1. The constant values can then be set for the required values.

6.1 AAXAA

Hexadecimal Calculation—Add, Subtract, Multiply, or Divide

Result

Hex value:
 $aa \times aa = 70E4$

Decimal value:
 $170 \times 170 = 28900$

= ?

Figure 4 AAXAA expected result

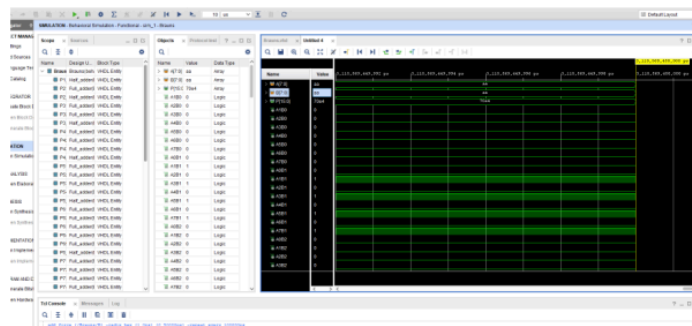


Figure 5 - AAXAA simulation

The simulation of AAXAA found in Figure 5 is identical to the expected result found in Figure 4. Both results showed the correct answer – 70E4.

6.2 FFXFF

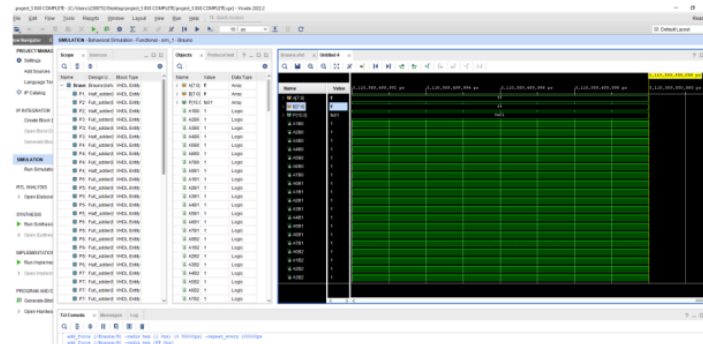


Figure 6 - Simulation results FFXFF

Hex Calculator
Hexadecimal Calculation—Add, Subtract, Multiply, or Divide

Result
Hex value:
FF × FF = **FE01**

Decimal value:
255 × 255 = **65025**

FF × FF = ?
Calculate

Convert Hexadecimal Value to Decimal Value
Hexadecimal Value: DAD = ?
Calculate

Convert Decimal Value to Hexadecimal Value
Decimal Value: 170 = ?

Math Calculators
Scientific Function
Percentage Trigonometric
Algebraic Standard Deviation
Geometry More Math Calculators
Financial Fitness and Health Math Other

Figure 7 -expected FFXFF

Figure 7 shows that the expected results of FFXFF should be FE01. Figure 6 Confirms the accuracy of my simulation, also achieving the same result.

6.3 11X11

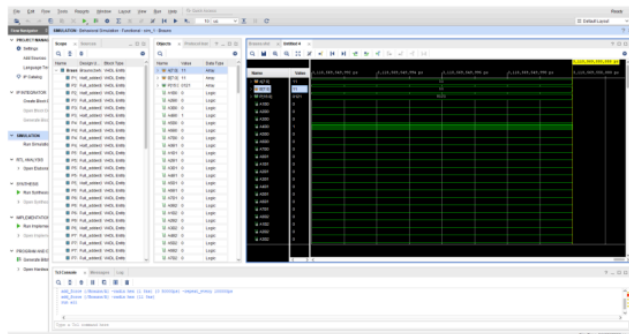


Figure 8 - 11x11 simulation

Hexadecimal Calculation—Add, Subtract, Multiply, or Divide

Result

Hex value:
 $11 \times 11 = 121$

Decimal value:
 $17 \times 17 = 289$

11 × 11 = ?

Calculate Clear

Convert Hexadecimal Value to Decimal Value

Figure 9 11x11 expected results

Carrying out the equation 11×11 , Figure 9 shows that the expected result will be 121. My simulation shown in Figure 8 confirms that my simulation is correct.

6.4 8AX55

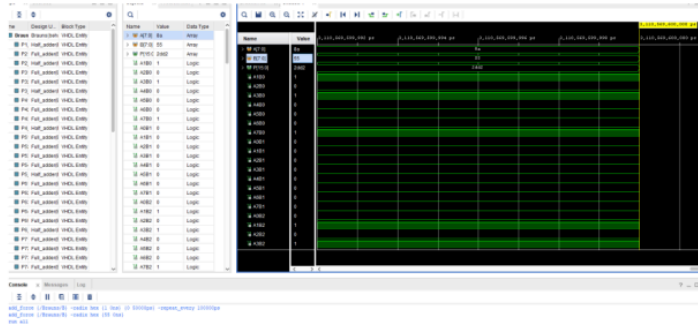


Figure 10 - 8ax55 simulation

Hexadecimal Calculation—Add, Subtract, Multiply, or Divide

Result

Hex value:

$8A \times 55 = 2DD2$

Decimal value:

$138 \times 85 = 11730$

Figure 11 - 8A x 55 expected results

Once again my simulation in Brauns multiplier was correct. Figure 11 shows the expected result of 8Ax55 to be 2DD2. My simulation in Figure 10 confirms this.

7. Wallace tree multiplier

7.1 FFxFF

Hex value:
 $ff \times ff = \text{FE01}$

Decimal value:
 $255 \times 255 = 65025$

= ?

Convert Hexadecimal Value to Decimal Value

Hexadecimal Value: = ?

[Math C](#)
[Scientific](#)
[Percenta](#)
[Volume](#)
[Random](#)
[Generato](#)

Figure 12 - FFxFF expected results

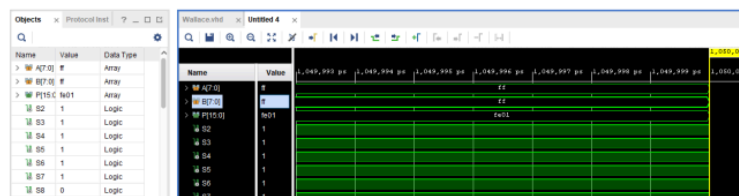


Figure 13 FFxFF Actual results

Using Wallace tree multiplier, I carried out another 4 multiplications to ensure that the coding worked correctly. As Figure 12 shows, the expected results of FFxFF should be FE01, my actual results seen in Figure 13 confirm this.

7.2 AAxAA

Hex Calculator

Hexadecimal Calculation—Add, Subtract, Multiply, or Divide

Result 

Hex value:
 $aa \times aa = 70E4$

Decimal value:
 $170 \times 170 = 28900$

= ?

Figure 14 - AAxAA expected results

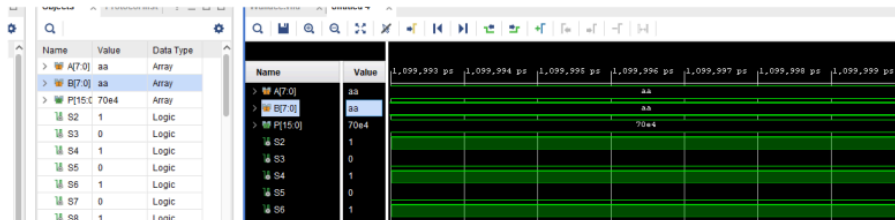


Figure 15 - Actual results

When carrying out the sum AAxAA, Figure 14 shows the answer to be 70E4. This is the same result using my Wallace tree multiplier seen in Figure 15.

7.3 11x23

Hex Calculator

Hexadecimal Calculation—Add, Subtract, Multiply, or Divide

Result

Hex value:

$11 \times 23 = 253$

Decimal value:

$17 \times 35 = 595$

11 x 23 = ?

Calculate Clear

Convert Hexadecimal Value to Decimal Value

Figure 16 - 11x23 Expected results

Data Type	Name	Value	1,147,993 px	1,147,994 px	1,147,995 px	1,147,996 px	1,147,997 px	1,147,998 px	1,147,999 px	1,148,000 px
Array	02 (0)	11								
Array	03 (0)	23								
Logic	04 (0)	0053								
Logic	05	0								
Logic	06	0								
Logic	07	1								
Logic	08	1								
Logic	09	0								

Figure 17 - 11x23 Actual results

8

To ensure clarity, I also carried out multiplications using smaller values, this aided in proving that the coding was correct throughout. Figure 16 shows the expected results to be 253, this matches with my actual results shown in Figure 17

7.4 41x53

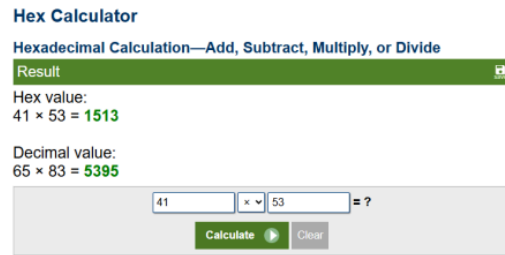


Figure 18 - 41x53 expected results

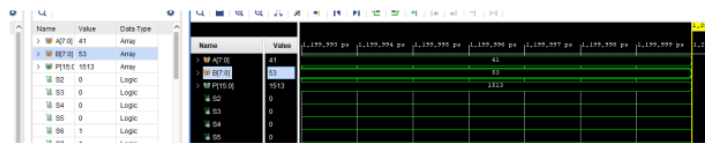


Figure 19 41x53 actual results

Finally, I carried out an equation in the middle of the spectrum, with 41x53 as an example. Figure 18 shows the expected results to be 1513. My actual results shown Figure 19 match this accurately.

7.5 Conclusion

My expected results and actual results matched correctly across all equations on both the Brauns multiplier and the Wallace tree multiplier. These simulations using the Vivado software proves that my coding was correct and they both function correctly with no issues.

The expected and actual results aligned consistently across all of the test cases with both the Brauns multiplier and the Wallace tree multiplier. These results, obtained via simulations in Vivado, confirm the correctness of my code and validate the functional accuracy of both different multiplier designs. There were no discrepancies or other issues seen during testing which shows the reliability of the implemented code.

8. Brauns and Wallace tree comparisons

8.1 Brauns

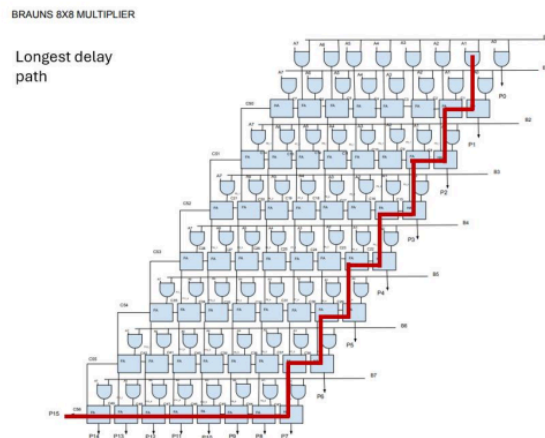


Figure 20 - Brauns Multiplier longest delay path

The longest delay path is shown in Figure 20. This is calculated using an AND gate delay of 1ns and a Full/half adder delay of 2ns. Following the longest delay path, this totals 41ns (20x Full or half adders and 1x AND gate delay).

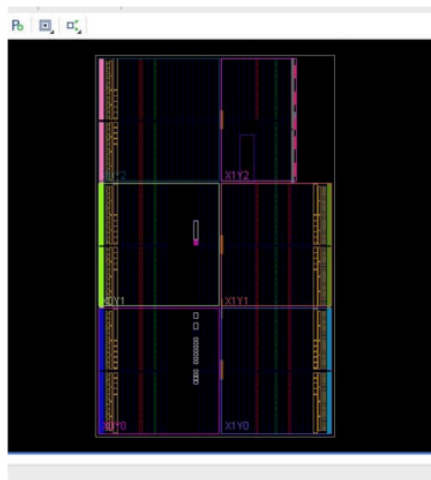


Figure 21

Figure 21 represents a graphical representation of how logic blocks are placed and routed within the FPGA. This is shown after carrying out the synthesis

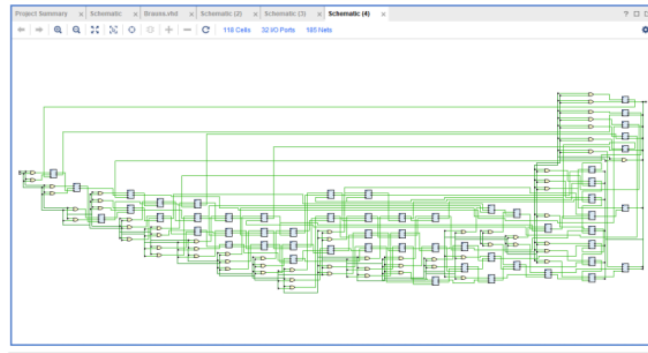


Figure 22 – RTL analysis

Figure 22 shows the schematic view of my Brauns design at the register transfer level (RTL). This can be generated after synthesis and before implementation. This image can help to verify that the design was synthesised as expected.

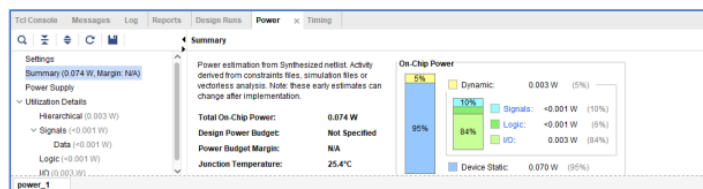


Figure 23 - Power usage

The power consumption of the brauns multiplier is shown in Figure 23. This shows power usage of 0.74kw

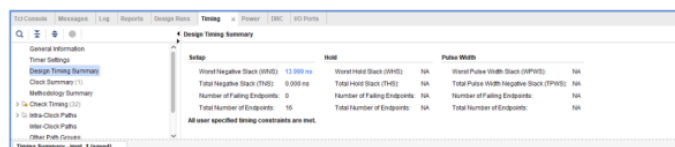


Figure 24 Timing summary

Figure 24 shows the overall timing summary of the worst negative stack. This value is measured at 13.999ns for Brauns multiplier

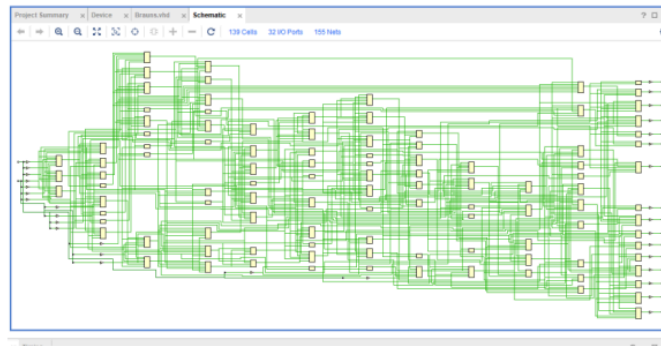


Figure 25

Figure 25 is a representation of Brauns multiplier schematic design

Pin	Function	Value
D-415	D-415	01 015
D-416	D-416	01 016
D-417	D-417	01 017
D-418	D-418	01 018
D-419	D-419	01 019
D-420	D-420	01 020
D-421	D-421	01 021
D-422	D-422	01 022
D-423	D-423	01 023
D-424	D-424	01 024
D-425	D-425	01 025
D-426	D-426	01 026
D-427	D-427	01 027
D-428	D-428	01 028
D-429	D-429	01 029
D-430	D-430	01 030
D-431	D-431	01 031
D-432	D-432	01 032
D-433	D-433	01 033
D-434	D-434	01 034
D-435	D-435	01 035
D-436	D-436	01 036
D-437	D-437	01 037
D-438	D-438	01 038
D-439	D-439	01 039
D-440	D-440	01 040
D-441	D-441	01 041
D-442	D-442	01 042
D-443	D-443	01 043
D-444	D-444	01 044
D-445	D-445	01 045
D-446	D-446	01 046
D-447	D-447	01 047
D-448	D-448	01 048
D-449	D-449	01 049
D-450	D-450	01 050
D-451	D-451	01 051
D-452	D-452	01 052
D-453	D-453	01 053
D-454	D-454	01 054
D-455	D-455	01 055
D-456	D-456	01 056
D-457	D-457	01 057
D-458	D-458	01 058
D-459	D-459	01 059
D-460	D-460	01 060
D-461	D-461	01 061
D-462	D-462	01 062
D-463	D-463	01 063
D-464	D-464	01 064
D-465	D-465	01 065
D-466	D-466	01 066
D-467	D-467	01 067
D-468	D-468	01 068
D-469	D-469	01 069
D-470	D-470	01 070
D-471	D-471	01 071
D-472	D-472	01 072
D-473	D-473	01 073
D-474	D-474	01 074
D-475	D-475	01 075
D-476	D-476	01 076
D-477	D-477	01 077
D-478	D-478	01 078
D-479	D-479	01 079
D-480	D-480	01 080
D-481	D-481	01 081
D-482	D-482	01 082
D-483	D-483	01 083
D-484	D-484	01 084
D-485	D-485	01 085
D-486	D-486	01 086
D-487	D-487	01 087
D-488	D-488	01 088
D-489	D-489	01 089
D-490	D-490	01 090
D-491	D-491	01 091
D-492	D-492	01 092
D-493	D-493	01 093
D-494	D-494	01 094
D-495	D-495	01 095
D-496	D-496	01 096
D-497	D-497	01 097
D-498	D-498	01 098
D-499	D-499	01 099
D-500	D-500	01 100

Figure 26 - Pin setting

Figure 26 displays how I set the pins to imitate a Basys 3 FPGA board

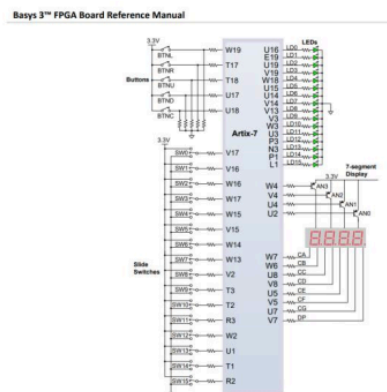


Figure 16. General purpose I/O devices on the Basyx 3.

Figure 27 - Pin requirements

The pins used in Figure 26 were taken from the Basys 3 schematic displayed in Figure 27

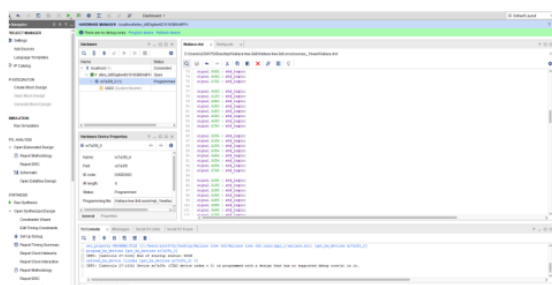


Figure 28 Timing

Figure 28 and Figure 29 is focused on the timing simulation/debugging phase after programming. It shows the timing behaviour across all of the pins.

```

1 create_clock -period 40.000 -name virtual_clk
2
3 # Set delays based on analytical calculations
4 set_input_delay -clock virtual_clk -max 1,000 [get_ports A*]
5 set_input_delay -clock virtual_clk -max 1,000 [get_ports B*]
6 set_output_delay -clock virtual_clk -max 2,000 [get_ports P*]
7
8 # Set max delay for the critical path (impact delay)
9 set_max_delay -from [get_ports A*] -to [get_ports P*] 35,000
10
11 net_property B3AG62G_F3J8 R2 [get_ports A[7]]
12 net_property B3AG62G_F3J8 R1 [get_ports A[8]]
13 net_property B3AG62G_F3J8 R0 [get_ports A[9]]
14 net_property B3AG62G_F3J8 K2 [get_ports A[4]]
15 net_property B3AG62G_F3J8 K3 [get_ports A[3]]
16 net_property B3AG62G_F3J8 T2 [get_ports A[2]]
17 net_property B3AG62G_F3J8 T3 [get_ports A[1]]
18 net_property B3AG62G_F3J8 V2 [get_ports A[0]]
19 net_property B3AG62G_F3J8 W3 [get_ports B[7]]
20 net_property B3AG62G_F3J8 W4 [get_ports B[6]]
21 net_property B3AG62G_F3J8 W5 [get_ports B[5]]
22 net_property B3AG62G_F3J8 W6 [get_ports B[4]]
23 net_property B3AG62G_F3J8 W7 [get_ports B[3]]
24 net_property B3AG62G_F3J8 W4 [get_ports B[2]]
25 net_property B3AG62G_F3J8 V6 [get_ports B[1]]
26 net_property B3AG62G_F3J8 V7 [get_ports B[0]]
27 net_property B3AG62G_F3J8 S1 [get_ports P[3]]
28 net_property B3AG62G_F3J8 S1 [get_ports P[14]]

```

Figure 29

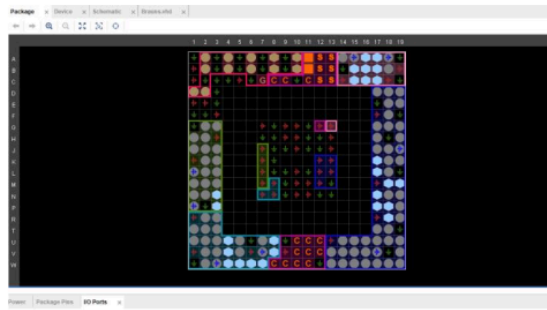


Figure 30

Figure 30 shows the i/o layout of brauns multiplier program. This shows how the design is mapped onto the FPGA fabric.

8.2 Wallace

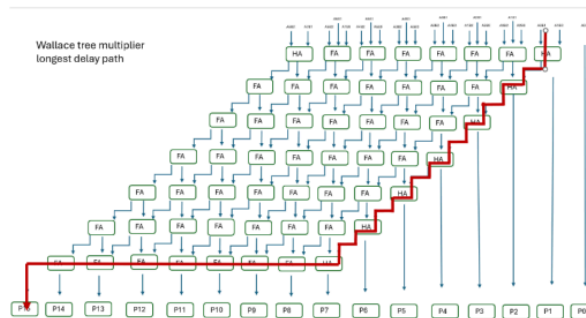


Figure 31 - Wallace tree longest delay timer

The longest delay of the Wallace tree multiplier is calculated using the same 10 gates as the Brauns. In this case, 14x Half/Full adders and 1 And gate = 29ns

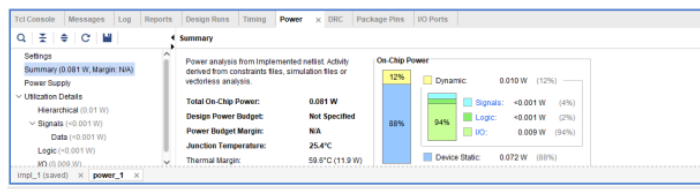


Figure 32 - Wallace tree power usage

Figure 32 shows the amount of power used in the Wallace tree multiplier. This is measured at 0.081W

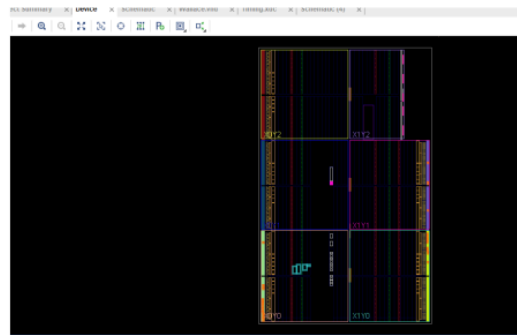


Figure 33

Figure 33 represents a graphical representation of how logic blocks are placed and routed within the FPGA



Figure 34

Figure 34 shows the i/o layout of the Wallace tree multiplier program. This shows how the design is mapped onto the FPGA fabric.

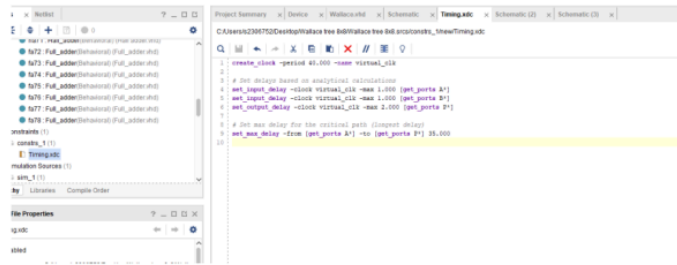


Figure 35

Figure 35 shows the parameters set for Wallace tree multiplier max clock speed period

Setup	Hold	Setup	Hold
Worst Negative Slack (NSD)	11.433 ns	Worst Positive Slack (PPSD)	NA
Total Negative Slack (NSD)	0.000 ns	Total Positive Slack (PPSD)	NA
Number of Failing Endpoints	0	Number of Failing Endpoints	NA
Total Number of Endpoints	10	Total Number of Endpoints	NA

Figure 36

The worst negative stack of Wallace tree multiplier is shown in Figure 36. This value is 11.43ns

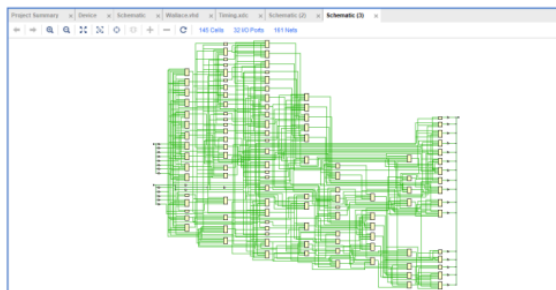


Figure 37

Figure 37 is a representation of Brauns multiplier schematic design

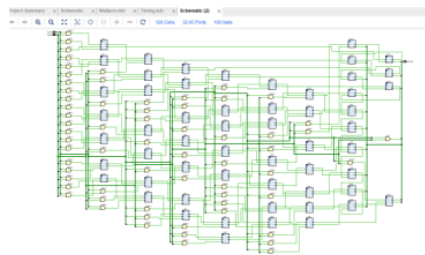


Figure 38

The pin layout was set in the Vivado software as shown in Figure 40. The pins were determined following the Basys 3 FPGA board reference manual (Figure 41).



Figure 41

8.3 Conclusion

The simulation that I have carried out using Brauns and Wallace tree multipliers have shown that the Wallace tree is the faster method for processing. The Wallace tree Multiplier was able to complete its worst negative stack in 11.4ns(Figure 36), whereas the Brauns multiplier would take 14ns for its worst negative stack (Figure 24).

The experiment confirms my initial statement that the Wallace tree multiplier is more energy intensive than Brauns. Wallace tree multiplier used 0.081w in my simulation of on-chip power (shown in Figure 32), compared to 0.074w shown in Figure 23.

With Both simulations connected to the Basys 3 board, I carried out the FFxFF equation with my program downloaded. The results matched both of my simulations that were carried out within the software

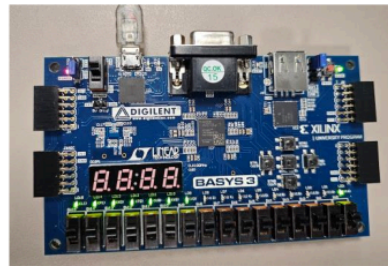


Figure 42 - Hardware test

The image above (Figure 42) shows the result of FFxFF with the correct LED light powered. The Answer is FE01, In Binary this would be F=all 4 lights on, E = first 3 lights on, 0 = 0 lights on and finally 1= just the light to the right on. This is how it successfully displays on the Basys 3 board.

Both Brauns and Wallace tree have proven to be suitable and efficient methods for multiplication operations. Brauns was much simpler to program and create and is suitable for more simple multiplication tasks. Wallace tree multiplier was more difficult and time consuming to code but it much more suited to simplifying more difficult multiplication processes, with the processing time much lower.

The longest delay path of both the Brauns and Wallace tree multipliers show that the Brauns multiplier (Figure 20) takes significantly longer to carry out the longest delay path compared to Wallace tree multiplier (Figure 31).

Both of the tasks have proven to have their advantages and disadvantages, with the type of task required an important factor in deciding which process you would use. Brauns multiplier is much more suited to smaller multiplication tasks, using less power, where Wallace tree multiplier is aimed at more difficult multiplications that require increased speed and performance, at the cost of increased power consumption.

References

16

- [1] P. M. Nair, "Design of 4-Bit Braun Multiplier," Fossee, 2015. [Online]. Available: file:///C:/Users/jan17/downloads/4_bit_Braun_multiplier_with_Kogge_Stone_Adder.pdf. [Accessed 01 04 2025].
- [2] A. R, "Braun's Multiplier Implementation using," 09 2011. [Online]. Available: <https://aircconline.com/vlsics/V2N3/2311vlsics17.pdf>. [Accessed 06 04 2025].
- [3] D. M. RAVI, "Design and Implementation of Wallace Tree Multiplier using Higher," 06 2016. [Online]. Available: <https://www.ijvdc.org/uploads/546213IJVDCS9984-85.pdf>. [Accessed 04 2025].
- [4] M. GUDHIMETLA, "COMPARISON OF DIFFERENT TYPES OF MULTIPLIERS WITH," 2020. [Online]. Available: https://www.digitalxplore.org/up_proc/pdf/315-150546881948-53.pdf. [Accessed 03 04 2025].
- [5] P. Kaushal, "VHDL - Understanding the Hardware Description Language," 07 05 2022. [Online]. Available: <https://www.copperpodip.com/post/vhdl-understanding-the-hardware-description-language>. [Accessed 06 04 2025].
- [6] "Hexidecimal calculator," [Online]. Available: <http://calculator.net/hex-calculator.html>. [Accessed 2025].

Chris James Electronics 2 assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

65/100

PAGE 1

PAGE 2

PAGE 3



Comment 1 | Knowledge & Understanding(20)

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



Comment 2 | Knowledge & Understanding(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 3 | Engagement with literature(20)

The report includes both multipliers and gives a good start on explaining their design and logic.

PAGE 4



Comment 4 | Engagement with literature(20)

The report makes an effort to refer to sources .However, adding more recent papers and doing a proper review of existing multiplier methods will improve the quality.



Comment 5 | Knowledge & Understanding(20)

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



Comment 6 | Cognitive skills (20)

Wallace
multiplier's dot diagram is neat.

PAGE 26

PAGE 27

PAGE 28

PAGE 29



Comment 7 | Practical Application (25)

Give
more detail about port mapping logic.

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36



Comment 8 | Practical Application (25)

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

PAGE 37

PAGE 38

PAGE 39



Comment 9 | Cognitive skills (20)

The report includes critical path diagrams and power analysis screenshots; however, LUT usage data and area details are missing.

PAGE 40

PAGE 41

PAGE 42



Comment 10 | Knowledge & Understanding(20)

Delay
diagrams and critical path calculations for both designs are shown clearly.

PAGE 43

PAGE 44



Comment 11 | Practical Application (25)

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

PAGE 45



Comment 12 | Practical Application (25)

The
report mentions pin mapping in the design flow, indicating awareness of
hardware setup.

PAGE 46



Comment 13 | Cognitive skills (20)

Basic
comparison of Braun and Wallace multipliers is included for speed, power and
area.



Comment 14 | Cognitive skills (20)

To
improve, the report should clearly interpret what these results mean in terms
of efficiency, performance, and suitability of each multiplier design.



Comment 15 | Cognitive skills (20)

It
would be useful if the report includes a well-defined section on future
recommendations.

PAGE 47



Comment 16 | Engagement with literature(20)

Not
many references are used. . Adding
more recent papers and doing a proper review of existing multiplier methods
will improve the quality.



Comment 17 | Transferable skills (15)

The
report is arranged nicely with clear sections.

GRADING FORM: UNTITLED

CHRIS JAMES

51.90

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.



Few relevant references are used, and they are less integrated. Analysis lacks originality or critical thought. Research gaps are not clearly identified. Minimal variety in sources.

9.5

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 concepts are needed. Component usage, and architecture explanations are mostly accurate, but some technical details (e.g., slack or LUTs) may lack depth. Applies general engineering knowledge with limited reference to broader context.

12

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



Good level of analysis shown. Partial product logic is missing. Architectural comparisons are mostly correct but needs more clarity. Conclusions are reasonable, though the conclusion should include numerical values on the performance of both multipliers.

12.7

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; Pin assignments are mostly correct. Simulation and hardware outputs are included with minor gaps. FPGA synthesis and report handling are well managed. Constraints are applied with general accuracy. Portmapping needs to be explained.

20.2

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.

10.2