

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

Electronics II

By Jay William Popham

B.Eng (Hons) in Electrical And Electronic Engineering
Level 6

Jenifer Prince Devaraj

University of Wales Trinity Saint David

Swansea, Wales

15/1/2024

Contents

1. introduction	4
2. Background research	4
2.1 Wallace Multiplier Background	4
2.2 Brauns Multiplier Background	5
2.3 VHDL	5
2.3.1 VHDL with public availability	5
2.3.2 VHDL design methodology and technology support	5
2.3.2 VHDL technology process independence	5
2.3.3 VHDL's wide range of descriptive capability	6
2.4 FPGA.....	6
3.Methodologies	6
3.1 Wallace Multiplier	6
3.2 Brauns Multiplier	7
5. A flowchart/pseudocode showing the system operation	8
5.1 Wallace Flowchart	8
5.2 Brauns Flowchart	8
6. VHDL Codes	9
6.1 Wallace VHDL Codes	9
6.2 Brauns VHDL	19
7. Simulation Results	31
7.1 Wallace Simulation	31
7.2 Brauns Simulation.....	35
8. Details of testing and results	39
8.1 Wallace	39
8.2 Brauns	41
8.3 Longest delay path.....	44
8.3.1 Wallace longest delay path	44
8.3.1 Brauns longest path delay.....	45
9. Conclusion	46
Bibliography	46

Table Of Figures

Figure 1.....	7
Figure 2.....	8
Figure 3.....	8
Figure 4.....	31
Figure 5.....	31
Figure 6.....	32
Figure 7.....	32
Figure 8.....	33
Figure 9.....	33
Figure 10.....	34
Figure 11.....	34
Figure 12.....	35
Figure 13.....	35
Figure 14.....	36
Figure 15.....	36
Figure 16.....	37
Figure 17.....	37
Figure 18.....	38
Figure 19.....	38
Figure 20.....	39
Figure 21.....	39
Figure 22.....	40
Figure 23.....	40
Figure 24.....	41
Figure 25.....	41
Figure 26.....	42
Figure 27.....	42
Figure 28.....	43
Figure 29.....	44
Figure 30.....	45

1. introduction

This report will cover the theoretical and practical aspects of Brauns and Wallace multipliers using very high-speed integrated circuit hardware descriptive language (VHDL). The report will involve background research of the multiplier application, expressing the capabilities, advantages, methodologies and application aspects of the Braun and Wallace style multipliers. The report will then carry on to a practical demonstration of how Braun's and Wallace multipliers are designed and critically evaluated. This report presents two different types of multipliers, each offering individual advantages in terms of speed, circuit complexity and power consumption.

2. Background research

Multiplication plays an important role in calculated operations and is used as a vital function in many applications such as DSP processors, microprocessors, image processing, and more. Multipliers are also used in a high portion of computing systems. The demand for high-speed processing has grown due to the expansion of signal processing applications. It is important to reduce time delay and power consumption for many digital circuit applications when trying to achieve a specific performance in real-time signal and image processing which rely on efficient calculated operations with higher output. Multiplication highly impacts the execution time of DSP algorithms, which will help when a high speed is needed, this is achieved by using adders at different stages and steps within the multiplier structure to achieve the desired operations. Full adders are commonly used in building multipliers, which are then simulated and analysed to gain improved outcomes while also minimizing the use of transistors in the overall implementation of multipliers, this then improves their usage in diverse digital circuits. The choice of a specific multiplier design depends on the application requirements. Multiplier styles can be categorized as "signed" and "unsigned" based on their characteristics. [1] [1]

2.1 Wallace Multiplier Background

The Wallace multiplier was first invented in 1964, it is a hardware implementation of digital circuits that multiplies two numbers. The Wallace multiplier can be split into a few different reduction approaches regarding speed and power, such as the conventional Wallace, the reduced complexity Wallace, a novel low power and high-speed multiplier, a booth recoded Wallace multiplier and an efficient high speed Wallace multiplier. The reduced complexity Wallace reduces around 80% of its half adders which then increases the number of full adders, this results in the difficulty of the Wallace multiplier being reduced in comparison to the conventional Wallace multiplier. The novel low power and high-speed Wallace tree multiplier has a lower power consumption and faster speed of up to 44% in comparison to the conventional multiplier. The Booth recoded Wallace tree multiplier has been proven to be 67% faster than the conventional Wallace multiplier, while the high-speed Wallace power consumption is reduced to 11% the speed also increases to 44% more than the conventional Wallace multiplier. [3]

2.2 Brauns Multiplier Background

The Braun's multiplier is part of the array multipliers group, it is also known as the carry save array Multiplier, the other type of array multiplier is known as the ripple-carry array. The array style is a digital combinational circuit which uses AND gates and full adders, applying this style usually involves the use of application specific integrated circuits (ASICs), which can be quite expensive to develop, therefore, it is important to thoroughly confirm the algorithms are correct before implementing them into a system. The ripple carry array is very slow when trying to deal with large implementations, this is due to the carry's having to go through every column in the row resulting in a longer critical path. On the other hand, the carry save array (Brauns) is used to solve the delayed critical path that would be produced by the ripple carry issue, this is because the carry can travel down to the columns instead of the columns being dependant to the columns to the right. The overall delay of the Braun's multiplier is influenced by both the delay of the full adders and the delay of the final adder in the last stage. [4]

2.3 VHDL

VHDL is a design language that helps the user of the language with complex designs. VHDL was invented to tackle problems that were recurring which were arising in the development and documentation of digital hardware. Prior to VHDL being invented there were no set standards in the industry for hardware descriptive language, the existing languages were normally made by for specific simulators that run the languages which tended to be developed by the companies who made them. VHDL offers many more capabilities and benefits over other hardware descriptive languages, such as public availability, design methodology, design technology support, technology and process independence and it has a wide range of descriptive capability. [3] [4]

2.3.1 VHDL with public availability

Public availability is one benefit when using VHDL, this is due to VHDL being set by the government as an IEEE standard. VHDL being set as an IEEE standard is a benefit due to it being a readable language by both machine and humans, this would then positively impact the implementation, synthesis and testing of hardware designs along with positively impacting the hardware design data and when hardware maintenance and modification is needed to be carried out. [3]

2.3.2 VHDL design methodology and technology support

Regarding the design methodology and technical support, this is one of the other benefits VHDL brings, this is delivered by being able to support different types of design methods as well as design technologies such as synchronous vs asynchronous or PLA vs random logic. These supporting aspects of the language will enable companies to gain assistance even if they have different design aspects or operate in a different way to others. VHDL is also compatible for library-based design tools such as CAD and CAE. [3]

2.3.2 VHDL technology process independence

VHDL also benefits users by being an independent regarding technology and the process of it, this is because it does not understand a single embedded system which results in the language being compatible when different implementations are used. [3]

2.3.3 VHDL's wide range of descriptive capability

VHDL also supports a wide range of descriptive capabilities, this is a benefit in comparison to other languages since VHDL can understand the behavioural description of the hardware when box level to gate level of a digital system is used. An advantage of VHDL is that it has an ability to capture many different descriptive levels at once, it can then simulate the system using many different levels of description, such as using an implementation of subsystems with high level behavioural descriptions of other subsystems. [3]

2.4 FPGA

FPGA stands for field-programmable gate array, it is a technology that offers advantages such as high hardware speed, parallelism, and software flexibility. FPGAs can be reprogrammed and adapt to changing requirements, making them more versatile unlike ASICs. FPGA has three main advantages when used, such as the architecture, the design techniques at the algorithmic level and the electronic design automation tools. FPGA tend to be used in applications such as DSP's, Embedded systems, telecommunications, aerospace and many more. [7]

3.Methodologies

Regarding the design of digital systems, multipliers are split into two subgroups, one of the groups are known as tree multipliers which the Wallace tree is developed from, and the second subgroups are known as array multipliers which is the group that the Brauns multiplier is from. The method that the tree multipliers use is by reducing the number of bits at each level in the tree until the calculation is complete. Array multipliers on the other hand use a fixed pattern to build their multipliers, resulting in compact designs and evenly distributed delays. The Tree multiplier approach can cause some delays in the design due to a complex tree structure. These delays are not evenly distributed and potentially result in glitches that consume additional power. The tree structure also requires a fair number of interconnections, resulting in a larger area covered. Despite these drawbacks, tree multipliers tend to consume less power compared to array multipliers. [5]

3.1 Wallace Multiplier

The Wallace multiplier is used to reduce the input conditions by grouping rows together reducing the size on each group. Rows that are not part of any group are then transferred to the next stage of the algorithm. The algorithm starts grouping each row into sets of three, a full adder will then be added for each group that contains three wires, a half adder will then be added for each group with two wires in, this then results in a set of partial products for the next stage of the algorithm. The steps then get repeated for any following columns that contain more than two bits or rows. [5]

The dot diagram below illustrates an 8X8 Wallace multiplier and how the Wallace algorithm is configured, by showing where the full and half adders are placed to create the partial products/algorithms.

Key:-

- Green = Half adders
- Blue = Full Adders

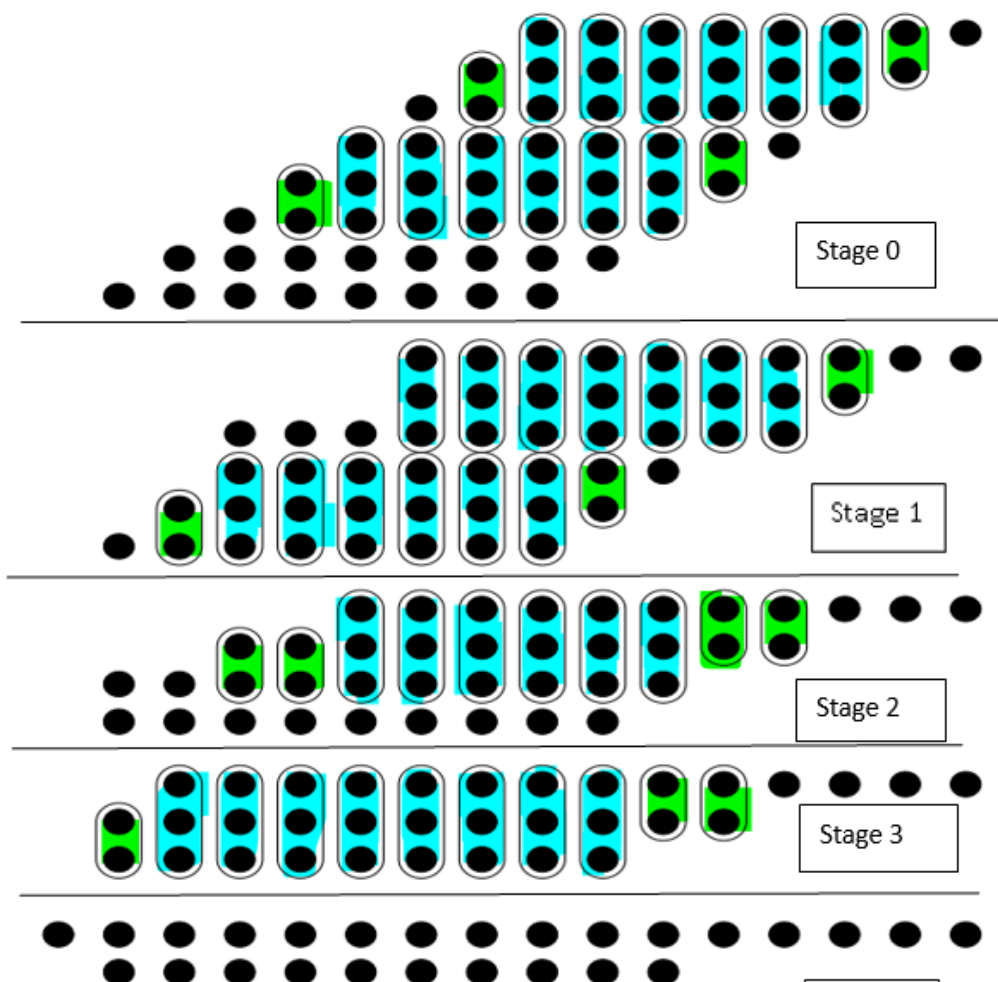


Figure 1

3.2 Brauns Multiplier

To execute a Brauns multiplier the approach is to multiply each bit in one row, with every bit in the other row. This is done for every bit in the first row, and then shifted position of the bit all the bits are then summed together and produce the result of the multiplication.

5. A flowchart/pseudocode showing the system operation

5.1 Wallace Flowchart

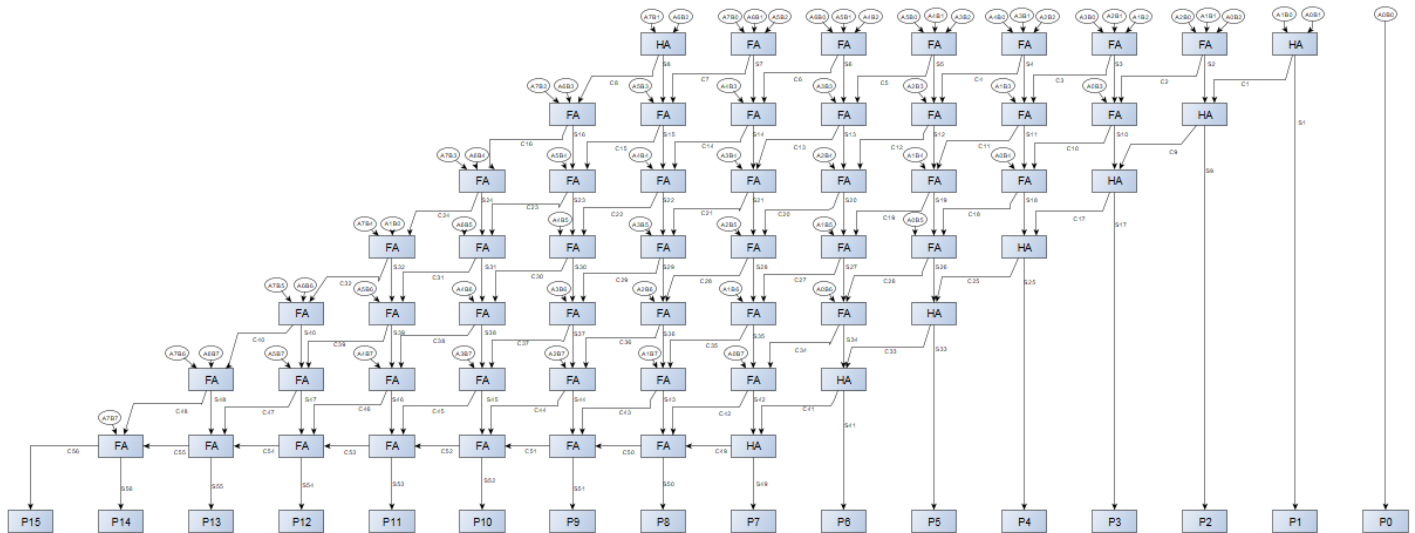


Figure 2

5.2 Brauns Flowchart

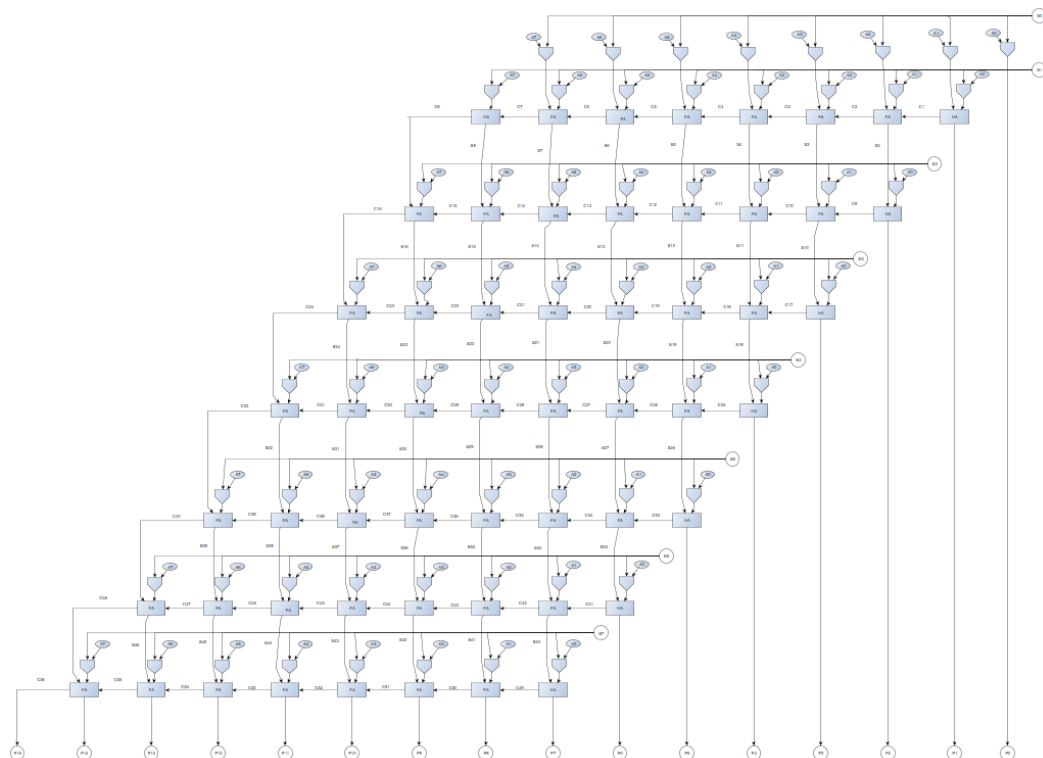


Figure 3

6. VHDL Codes

6.1 Wallace VHDL Codes

```
-----  
-- Company:  
-- Engineer:  
--  
-- Create Date: 07.12.2023 10:10:47  
-- Design Name:  
-- Module Name: wallace1 - 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;
```

```
-- 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 wallace1 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 wallace1;

architecture Behavioral of wallace1 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,S10,S11,S12,S13,S14,S15,S16,S18,S19,S20,S21,S22,S23,S24,S26,S27,S28,S29,S30
,S31,S32,S34,S35,S36,S37,S38,S39,S40,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(A1B0,A0B1,P(1),C1);

fa12 : Full_adder port map(A2B0,A1B1,A0B2,S2,C2);

fa13 : Full_adder port map(A3B0,A2B1,A1B2,S3,C3);

fa14 : Full_adder port map(A4B0,A3B1,A2B2,S4,C4);

```
fa15 : Full_adder port map(A5B0,A4B1,A3B2,S5,C5);
fa16 : Full_adder port map(A6B0,A5B1,A4B2,S6,C6);
fa17 : Full_adder port map(A7B0,A6B1,A5B2,S7,C7);
ha18 : Half_Adder port map(A7B1,A6B2,S8,C8);
```

--second stage

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

--third stage

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

--fourth stage

```
ha41 : Half_Adder port map(S18,C17,P(4),C25);
fa42 : Full_adder port map(S19,C18,A0B5,S26,C26);
fa43 : Full_adder port map(S20,C19,A1B5,S27,C27);
fa44 : Full_adder port map(S21,C20,A2B5,S28,C28);
fa45 : Full_adder port map(S22,C21,A3B5,S29,C29);
```

```
fa46 : Full_adder port map(S23,C22,A4B5,S30,C30);
fa47 : Full_adder port map(S24,C23,A5B5,S31,C31);
fa48 : Full_adder port map(A7B4,C24,A6B5,S32,C32);
```

--fifth stage

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

-- sixth stage

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

-- seventh stage

```
ha71 : Half_Adder port map(S42,C41,P(7),C49);
fa72 : Full_adder port map(S43,C42,C49,P(8),C50);
fa73 : Full_adder port map(S44,C43,C50,P(9),C51);
fa74 : Full_adder port map(S45,C44,C51,P(10),C52);
fa75 : Full_adder port map(S46,C45,C52,P(11),C53);
fa76 : Full_adder port map(S47,C46,C53,P(12),C54);
```

```
fa77 : Full_adder port map(S48,C46,C54,P(13),C55);  
fa78 : Full_adder port map(A7B7,C48,C55,P(14),P(15));  
  
end Behavioral;
```

6.2 Brauns VHDL

```
-- Company:
-- Engineer:
--
-- Create Date: 19.12.2023 20:07:44
-- Design Name:
-- Module Name: Braun - 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;

-- 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 BraunsMultiplier 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 BraunsMultiplier;
```

architecture Behavioral of BraunsMultiplier is

```
signal A1B0 : std_logic;  
signal A0B1 : std_logic;  
signal A2B0 : std_logic;  
signal A1B1 : std_logic;  
signal A0B2 : std_logic;  
signal A3B0 : std_logic;  
signal A2B1 : std_logic;  
signal A1B2 : std_logic;  
signal A0B3 : std_logic;  
signal A4B0 : std_logic;  
signal A3B1 : std_logic;  
signal A2B2 : std_logic;  
signal A1B3 : std_logic;  
signal A0B4 : std_logic;  
signal A5B0 : std_logic;  
signal A4B1 : std_logic;  
signal A3B2 : std_logic;  
signal A2B3 : std_logic;  
signal A1B4 : std_logic;  
signal A0B5 : std_logic;  
signal A6B0 : std_logic;  
signal A5B1 : std_logic;  
signal A4B2 : std_logic;  
signal A3B3 : std_logic;
```

```
signal A2B4 : std_logic;  
signal A1B5 : std_logic;  
signal A0B6 : std_logic;  
signal A7B0 : std_logic;  
signal A6B1 : std_logic;  
signal A5B2 : std_logic;  
signal A4B3 : std_logic;  
signal A3B4 : std_logic;  
signal A2B5 : std_logic;  
signal A1B6 : std_logic;  
signal A0B7 : std_logic;  
signal A7B1 : std_logic;  
signal A6B2 : std_logic;  
signal A5B3 : std_logic;  
signal A4B4 : std_logic;  
signal A3B5 : std_logic;  
signal A2B6 : std_logic;  
signal A1B7 : std_logic;  
signal A7B2 : std_logic;  
signal A6B3 : std_logic;  
signal A5B4 : std_logic;  
signal A4B5 : std_logic;  
signal A3B6 : std_logic;  
signal A2B7 : std_logic;  
signal A7B3 : std_logic;  
signal A6B4 : std_logic;  
signal A5B5 : std_logic;  
signal A4B6 : std_logic;  
signal A3B7 : std_logic;  
signal A7B4 : std_logic;  
signal A6B5 : std_logic;
```

```
signal A5B6 : std_logic;  
signal A4B7 : std_logic;  
signal A7B5 : std_logic;  
signal A6B6 : std_logic;  
signal A5B7 : std_logic;  
signal A7B6 : std_logic;  
signal A6B7 : std_logic;  
signal A7B7 : std_logic;
```

```
signal C1 : std_logic;--P2  
signal S2 : std_logic;
```

```
signal C2 : std_logic;--P3  
signal S3 : std_logic;  
signal C9 : std_logic;  
signal S10 : std_logic;
```

```
signal C3 : std_logic;--P4  
signal S4 : std_logic;  
signal C10 : std_logic;  
signal S11 : std_logic;  
signal C17 : std_logic;  
signal S18 : std_logic;
```

```
signal C4 : std_logic;--P5  
signal S5 : std_logic;  
signal C11 : std_logic;  
signal S12 : std_logic;  
signal C18 : std_logic;  
signal S19 : std_logic;
```

signal C25 : std_logic;

signal S26 : std_logic;

signal C5 : std_logic;--P6

signal S6 : std_logic;

signal C12 : std_logic;

signal S13 : std_logic;

signal C19 : std_logic;

signal S20 : std_logic;

signal C26 : std_logic;

signal S27 : std_logic;

signal C33 : std_logic;

signal S34 : std_logic;

signal C6 : std_logic;--P7

signal S7 : std_logic;

signal C13 : std_logic;

signal S14 : std_logic;

signal C20 : std_logic;

signal S21 : std_logic;

signal C27 : std_logic;

signal S28 : std_logic;

signal C34 : std_logic;

signal S35 : std_logic;

signal C41 : std_logic;

signal S42 : std_logic;

signal C7 : std_logic;--P8

signal S8 : std_logic;

signal C14 : std_logic;

signal S15 : std_logic;

```
signal C21 : std_logic;  
signal S22 : std_logic;  
signal C28 : std_logic;  
signal S29 : std_logic;  
signal C35 : std_logic;  
signal S36 : std_logic;  
signal C42 : std_logic;  
signal S43 : std_logic;  
signal C49 : std_logic;
```

```
signal C8 : std_logic;--P9  
signal C15 : std_logic;  
signal S16 : std_logic;  
signal C22 : std_logic;  
signal S23 : std_logic;  
signal C29 : std_logic;  
signal S30 : std_logic;  
signal C36 : std_logic;  
signal S37 : std_logic;  
signal C43 : std_logic;  
signal S44 : std_logic;  
signal C50 : std_logic;
```

```
signal C16 : std_logic;--P10  
signal C23 : std_logic;  
signal S24 : std_logic;  
signal C30 : std_logic;  
signal S31 : std_logic;  
signal C37 : std_logic;  
signal S38 : std_logic;  
signal C44 : std_logic;
```

signal S45 : std_logic;

signal C51 : std_logic;

signal C24 : std_logic;--P11

signal C31 : std_logic;

signal S32 : std_logic;

signal C38 : std_logic;

signal S39 : std_logic;

signal C45 : std_logic;

signal S46 : std_logic;

signal C52 : std_logic;

signal C32 : std_logic;--P12

signal C39 : std_logic;

signal S40 : std_logic;

signal C46 : std_logic;

signal S47 : std_logic;

signal C53 : std_logic;

signal C40 : std_logic;--P13

signal C47 : std_logic;

signal S48 : std_logic;

signal C54 : std_logic;

signal C48 : std_logic;--P14

signal C55 : std_logic;

signal C56 : std_logic;--p15

component Fulladder 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 Halfadder is
```

```
Port ( A : in STD_LOGIC;  
      B : in STD_LOGIC;  
      S : out STD_LOGIC;  
      COUT : out STD_LOGIC);
```

```
end component;
```

```
begin
```

```
A1B0 <= A(1) and B(0);  
A0B1 <= A(0) and B(1);  
A2B0 <= A(2) and B(0);  
A1B1 <= A(1) and B(1);  
A0B2 <= A(0) and B(2);  
A3B0 <= A(3) and B(0);  
A2B1 <= A(2) and B(1);  
A1B2 <= A(1) and B(2);  
A0B3 <= A(0) and B(3);  
A4B0 <= A(4) and B(0);  
A3B1 <= A(3) and B(1);  
A2B2 <= A(2) and B(2);  
A1B3 <= A(1) and B(3);  
A0B4 <= A(0) and B(4);  
A5B0 <= A(5) and B(0);
```

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

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

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

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

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

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

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

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

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

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

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

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

A0B6 <= A(0) and B(6);

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

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

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

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

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

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

A1B6 <= A(1) and B(6);

A0B7 <= A(0) and B(7);

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

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

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

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

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

A2B6 <= A(2) and B(6);

A1B7 <= A(1) and B(7);

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

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

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

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

A3B6 <= A(3) and B(6);

A2B7 <= A(2) and B(7);

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

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

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

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

A3B7 <= A(3) and B(7);

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

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

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

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

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

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

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

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

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

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

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

P1 : Halfadder port map (A1B0 , A0B1 , P(1) , C1);--P1

S2_1 : Fulladder port map (A2B0 , A1B1 , C1, S2 , C2);--P2

P2: Halfadder port map (A0B2 , S2 , P(2) , C9);

S3_1 : Fulladder port map (A3B0 , A2B1 , C2 , S3 , C3);--P3

S10_1 : Fulladder port map (A1B2 , S3 , C9 , S10 , C10);

P3 : Halfadder port map (A0B3 , S10 , P(3) , C17);

S4_1 : Fulladder port map (A4B0 , A3B1 , C3 , S4 , C4);--P4

S11_1 : Fulladder port map (A2B2 , S4 , C10 , S11 , C11);

S18_1 : Fulladder port map (A1B3 , S11 , C17 , S18 , C18);

P4 : Halfadder port map (A0B4 , S18 , P(4) , C25);

S5_1 : Fulladder port map (A5B0 , A4B1 , C4 , S5 , C5);--P5

S12_1 : Fulladder port map (A3B2 , S5 , C11 , S12 , C12);

S19_1 : Fulladder port map (A2B3 , S12 , C18 , S19 , C19);

S26_1 : Fulladder port map (A1B4 , S19 , C25 , S26 , C26);

P5 : Halfadder port map (A0B5 , S26 , P(5) , C33);

S6_1 : Fulladder port map (A6B0 , A5B1 , C5 , S6 , C6);--P6

S13_1 : Fulladder port map (A4B2 , S6 , C12 , S13 , C13);

S20_1 : Fulladder port map (A3B3 , S13 , C19 , S20 , C20);

S27_1 : Fulladder port map (A2B4 , S20 , C26 , S27 , C27);

S34_1 : Fulladder port map (A1B5 , S27 , C33 , S34 , C34);

P6 : Halfadder port map (A0B6 , S34 , P(6) , C41);

S7_1 : Fulladder port map (A7B0 , A6B1 , C6 , S7 , C7);--P7

S14_1 : Fulladder port map (A5B2 , S7 , C13 , S14 , C14);

S21_1 : Fulladder port map (A4B3 , S14 , C20 , S21 , C21);

S28_1 : Fulladder port map (A3B4 , S21 , C27 , S28 , C28);

S35_1 : Fulladder port map (A2B5 , S28 , C34 , S35 , C35);

S42_1 : Fulladder port map (A1B6 , S35 , C41 , S42 , C42);

P7 : Halfadder port map (A0B7 , S42 , P(7) , C49);

S8_1 : Halfadder port map (A7B1 , C7 , S8 , C8);--P8

S15_1 : Fulladder port map (A6B2 , S8 , C14 , S15 , C15);

S22_1 : Fulladder port map (A5B3 , S15 , C21 , S22 , C22);

S29_1 : Fulladder port map (A4B4 , S22 , C28 , S29 , C29);

S36_1 : Fulladder port map (A3B5 , S29 , C35 , S36 , C36);

S43_1 : Fulladder port map (A2B6 , S36 , C42 , S43 , C43);

P8 : Fulladder port map (A1B7 , S43 , C49 , P(8) , C50);

S16_1 : Fulladder port map (A7B2 , C8 , C15 , S16 , C16);--P9

S23_1 : Fulladder port map (A6B3 , S16 , C22 , S23 , C23);

S30_1 : Fulladder port map (A5B4 , S23 , C29 , S30 , C30);

S37_1 : Fulladder port map (A4B5 , S30 , C36 , S37 , C37);

S44_1 : Fulladder port map (A3B6 , S37 , C43 , S44 , C44);

P9 : Fulladder port map (A2B7 , S44 , C50 , P(9) , C51);

S24_1 : Fulladder port map (A7B3 , C16 , C23 , S24 , C24);--P10

S31_1 : Fulladder port map (A6B4 , S24 , C30 , S31 , C31);

S38_1 : Fulladder port map (A5B5 , S31 , C37 , S38 , C38);

S45_1 : Fulladder port map (A4B6 , S38 , C44 , S45 , C45);

P10 : Fulladder port map (A3B7 , S45 , C51 , P(10) , C52);

S32_1 : Fulladder port map (A7B4 , C24 , C31 , S32 , C32);--P11

S39_1 : Fulladder port map (A6B5 , S32 , C38 , S39 , C39);

S46_1 : Fulladder port map (A5B6 , S39 , C45 , S46 , C46);

P11 : Fulladder port map (A4B7 , S46 , C52 , P(11) , C53);

S40_1 : Fulladder port map (A7B5 , C32 , C39 , S40 , C40);--P12

S47_1 : Fulladder port map (A6B6 , S40 , C46 , S47 , C47);

P12 : Fulladder port map (A5B7 , S47 , C53 , P(12) , C54);

S48_1 : Fulladder port map (A7B6 , C40 , C47 , S48 , C48);--P13

P13 : Fulladder port map (A6B7 , S48 , C54 , P(13) , C55);

P14 : Fulladder port map (A7B7 , C48 , C55 , P(14) , C56);--P14

P15 : P(15) <= C56;--P15

end Behavioral;

7. Simulation Results

7.1 Wallace Simulation

The figure below illustrates the behavioural simulation being run which leads onto the timing diagram, the clock is forced for both A & B with a leading-edge value of 0 and a trailing edge value of 0. The constant for each variable is then set, A=9 & B=8. The value produced is 48, proving that the written code is working correctly.

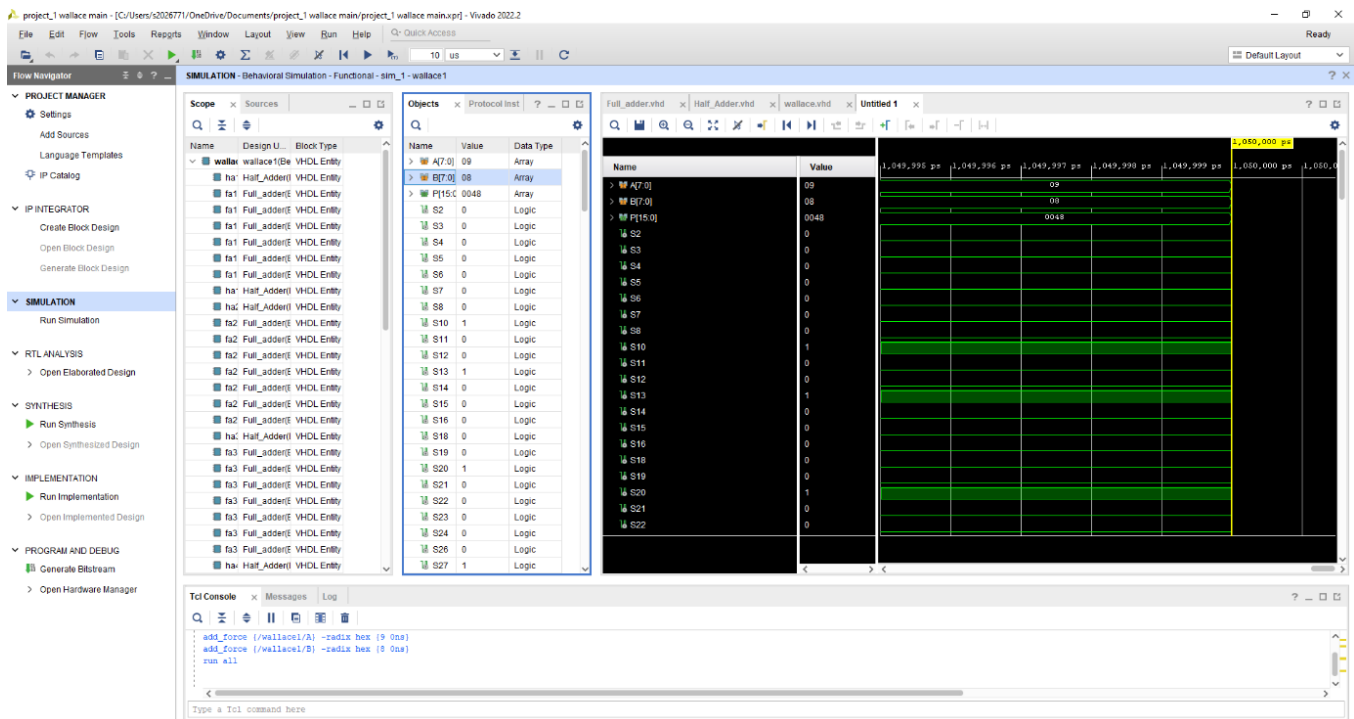


Figure 4

The figure below illustrates the next step, which is to run the synthesis, as shown in the top right-hand corner the synthesis is complete.

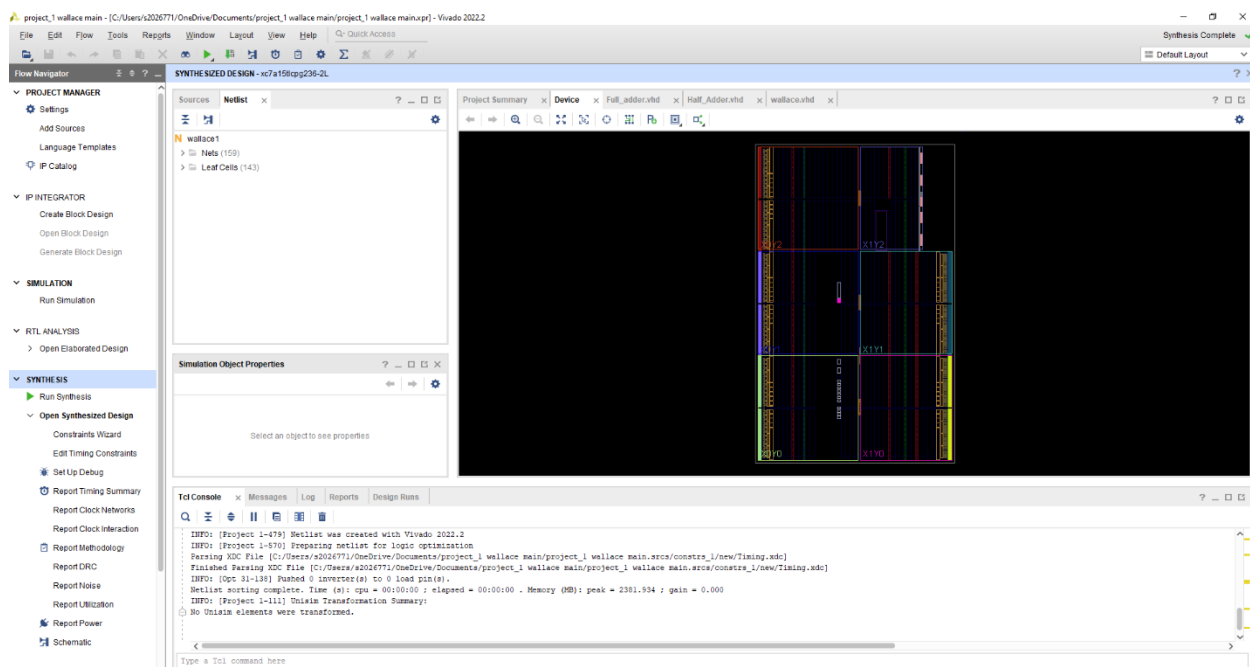


Figure 5

The next image illustrates the schematic diagram.

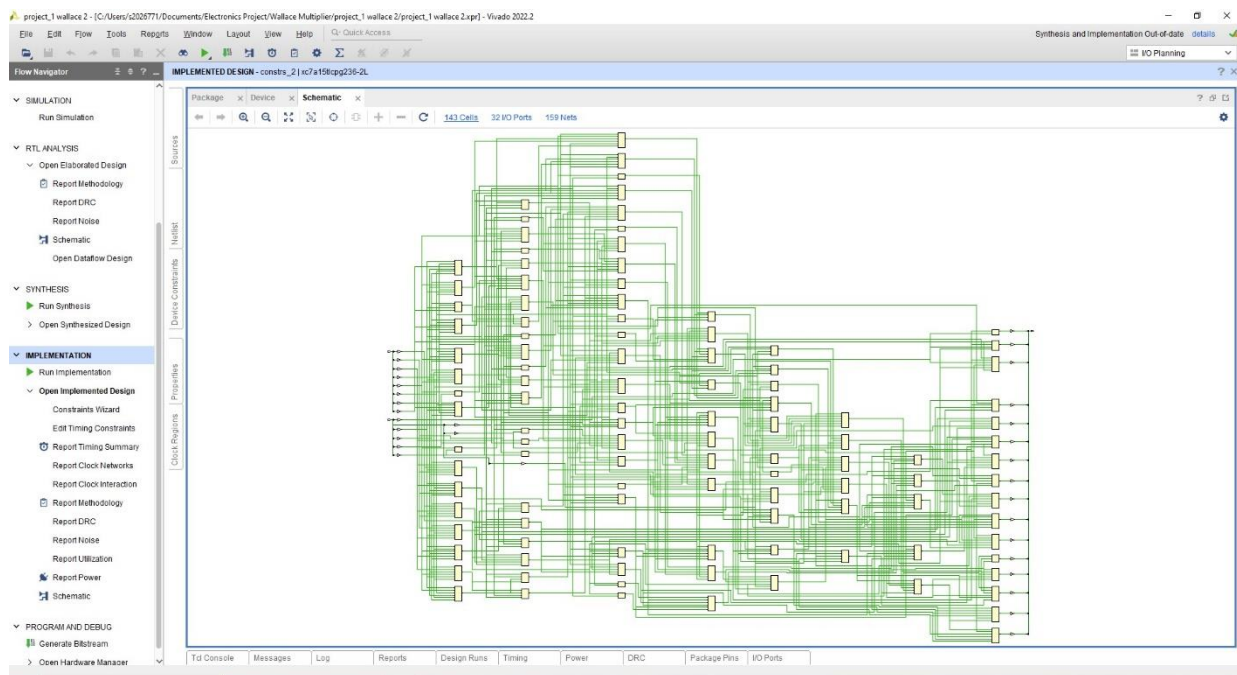


Figure 6

The below illustrates all the I/O ports being assigned to their pins.

The screenshot shows the 'SYNTHESIZED DESIGN' view in Vivado 2022.2. The main workspace displays a table of I/O ports and their assignments to pins. The table has the following columns: Name, Direction, Interface, Neg Diff Pair, Package Pin, Fixed, Bank, I/O Std, Vcco, Vref, Drive Strength, Slew Type, Pull Type, Off-Chip Termination, IN_TERM, and Partition Pin Location. The table lists 32 I/O ports, each with its corresponding pin assignment and configuration details.

Name	Direction	Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM	Partition Pin Location
A[7]	IN			A4	✓	215									N/A
A[5]	IN			A5	✓	215									N/A
A[5]	IN			A14	✓	16	default (LVCMOS18)	1.800				NONE	NONE	✓	N/A
A[4]	IN			A15	✓	16	default (LVCMOS18)	1.800				NONE	NONE	✓	N/A
A[3]	IN			A16	✓	16	default (LVCMOS18)	1.800				NONE	NONE	✓	N/A
A[2]	IN			A17	✓	16	default (LVCMOS18)	1.800				NONE	NONE	✓	N/A
A[1]	IN			A18	✓	16	default (LVCMOS18)	1.800				NONE	NONE	✓	N/A
A[0]	IN			B4	✓	215									N/A
B[7]	IN			B6	✓	215									N/A
B[5]	IN			B8	✓	215									N/A
B[5]	IN			B10	✓	215									N/A
B[4]	IN			B15	✓	16	default (LVCMOS18)	1.800				NONE	NONE	✓	N/A
B[3]	IN			B16	✓	16	default (LVCMOS18)	1.800				NONE	NONE	✓	N/A
B[2]	IN			B17	✓	16	default (LVCMOS18)	1.800				NONE	NONE	✓	N/A
B[1]	IN			B18	✓	16	default (LVCMOS18)	1.800				NONE	NONE	✓	N/A
B[0]	IN			C15	✓	16	default (LVCMOS18)	1.800				NONE	NONE	✓	N/A
P[15]	OUT			C16	✓	16	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[14]	OUT			C17	✓	16	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[13]	OUT			D17	✓	14	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[12]	OUT			D18	✓	14	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[11]	OUT			D19	✓	14	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[10]	OUT			E18	✓	14	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[9]	OUT			F18	✓	14	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[8]	OUT			G2	✓	35	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[7]	OUT			G3	✓	35	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[6]	OUT			G17	✓	14	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[5]	OUT			G18	✓	14	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[4]	OUT			G19	✓	14	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[3]	OUT			H1	✓	35	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[2]	OUT			H2	✓	35	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[1]	OUT			H17	✓	14	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A
P[0]	OUT			H19	✓	14	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50	✓	N/A

Figure 7

The next figure demonstrates the implementation being run and complete as shown in the top right-hand corner of the image.

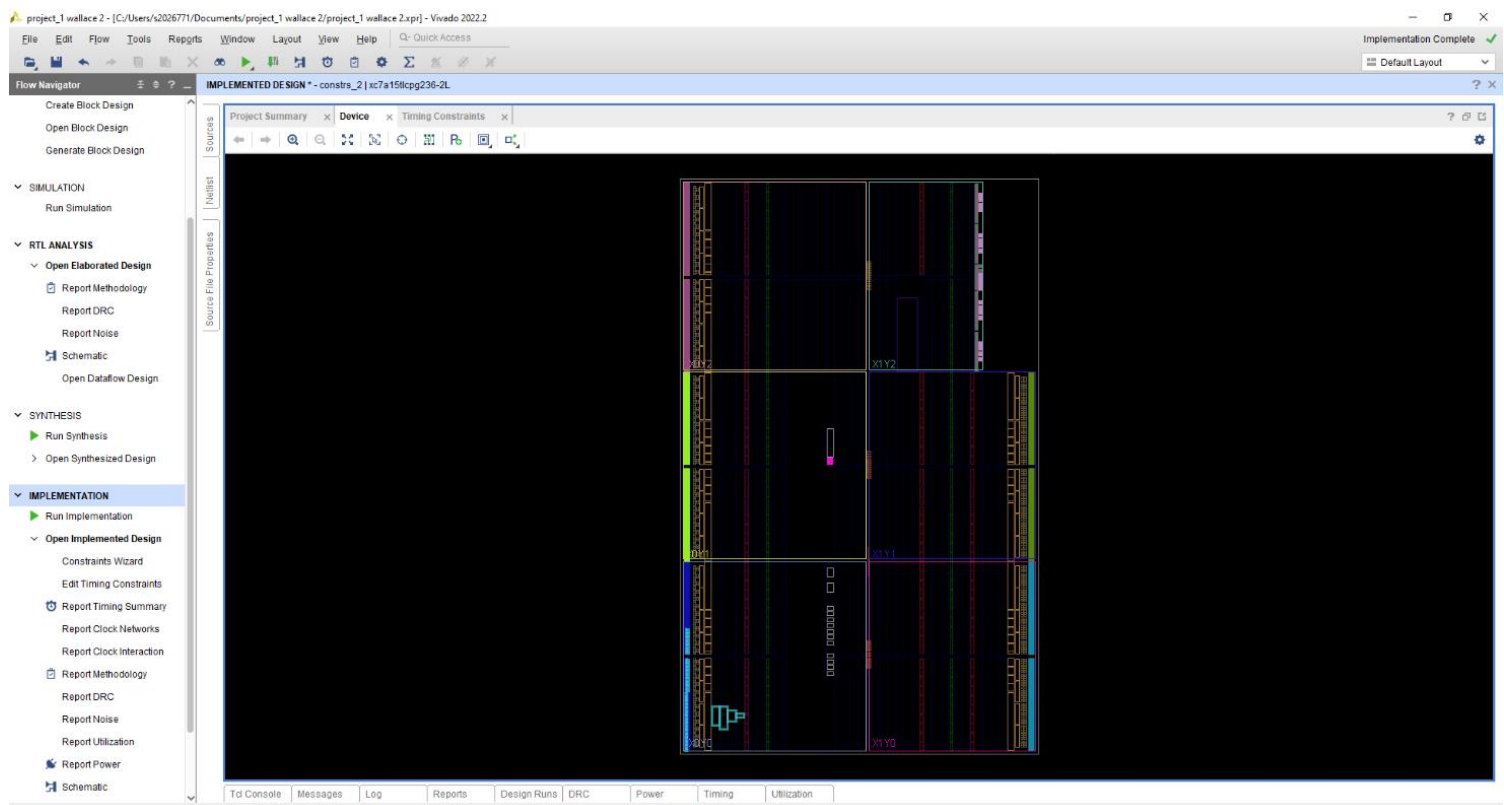


Figure 8

The next figure illustrates the timing report summary, this shows that the worst negative slack is reacting faster than normal, the signal is arriving early resulting in wasting area and power by being unnecessarily fast, a timing restraint will need to get added to ensure that no signal is missing the timing. The total negative slack is at infinity.

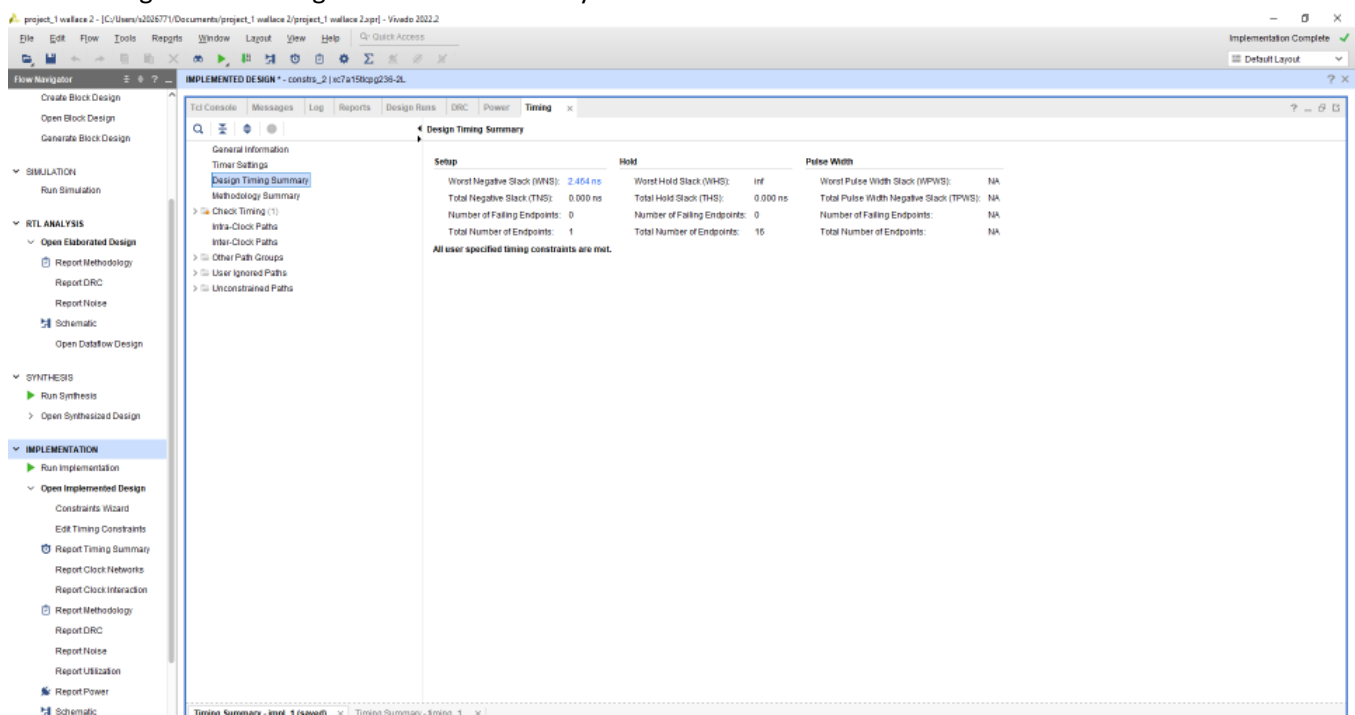


Figure 9

The figure that follows demonstrates the timing constraints, the clock value is set then the longest delay of negative slack is added as a time constraint, this then ensures that no signal is missing the timing.

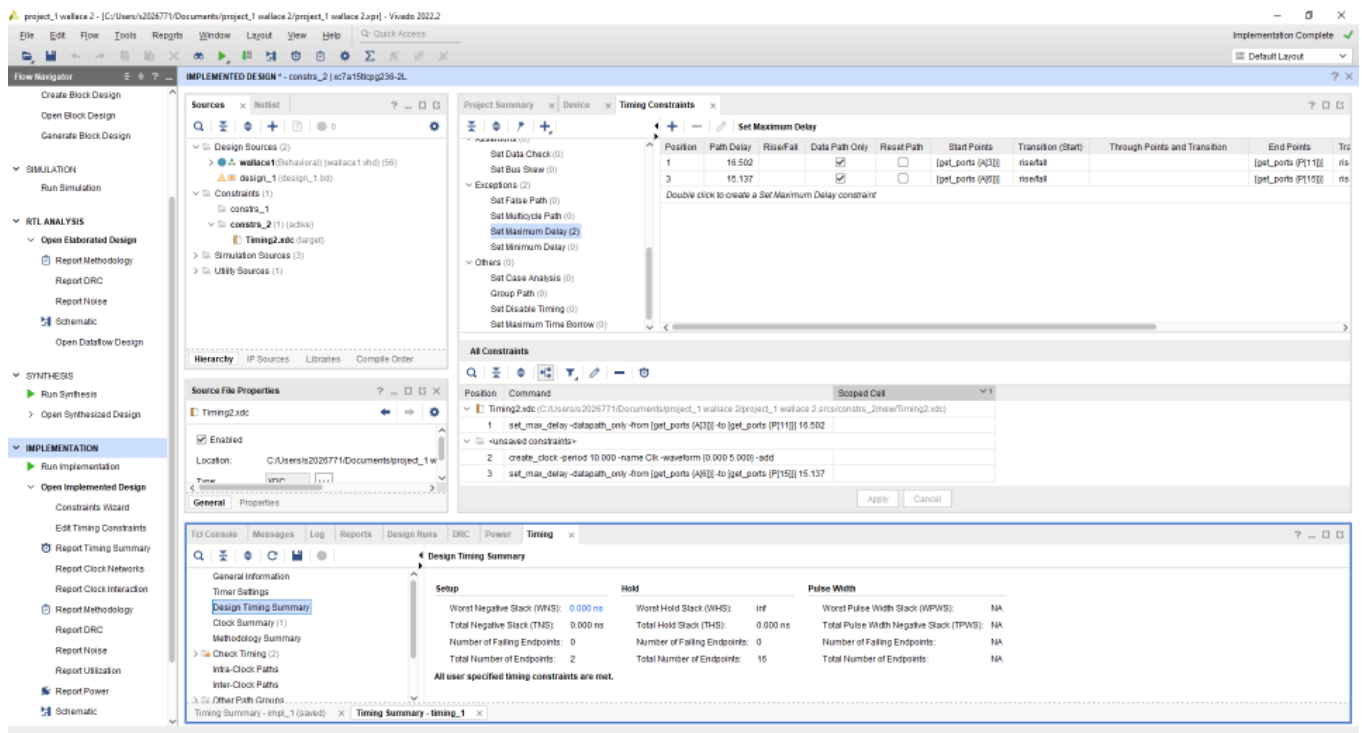


Figure 10

The image below illustrates the I/O Planning of the Wallace multiplier.

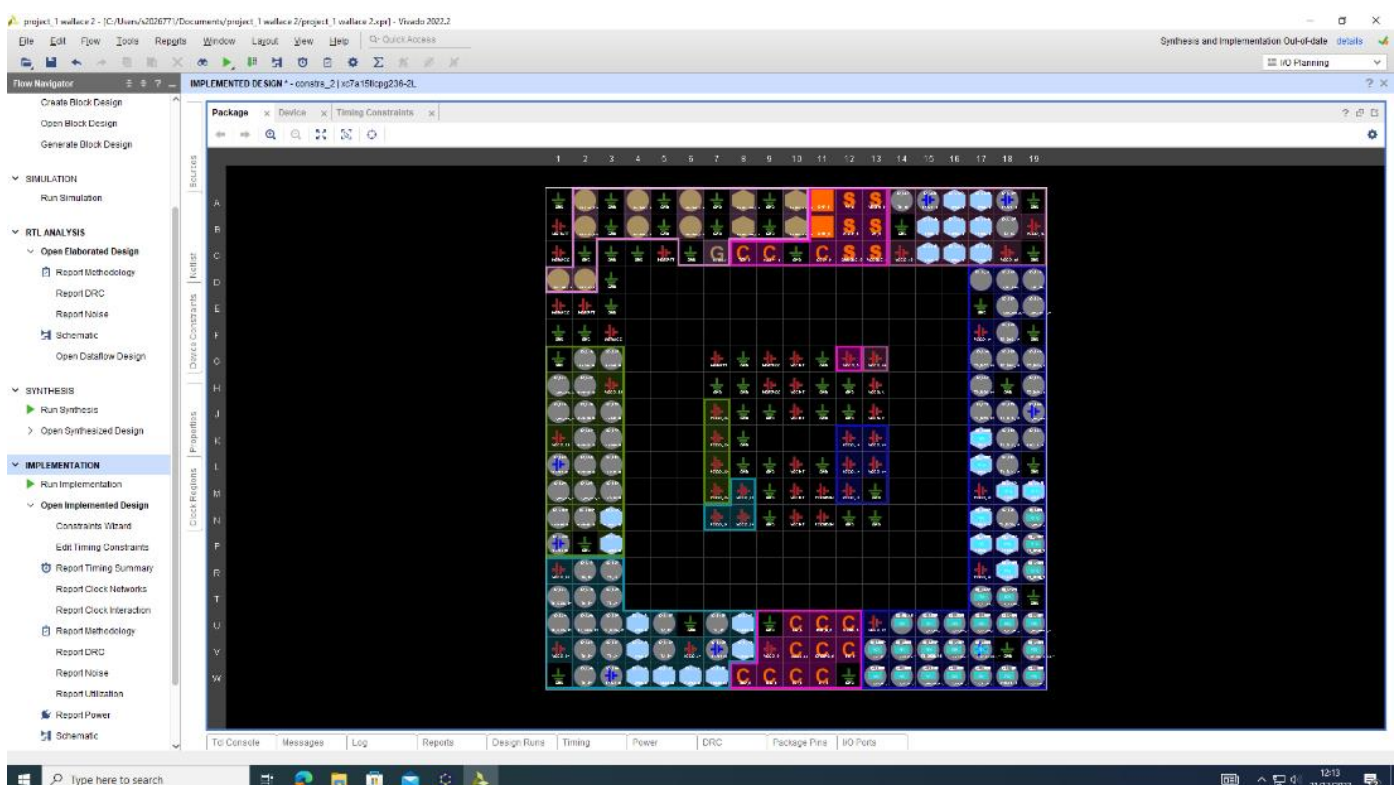


Figure 11

7.2 Brauns Simulation

The figure below illustrates the behavioural simulation being run which leads onto the timing diagram, the clock is forced for both A & B with a leading-edge value of 0 and a trailing edge value of 0. The constant for each variable is then set at different values each time with the clock staying the same.

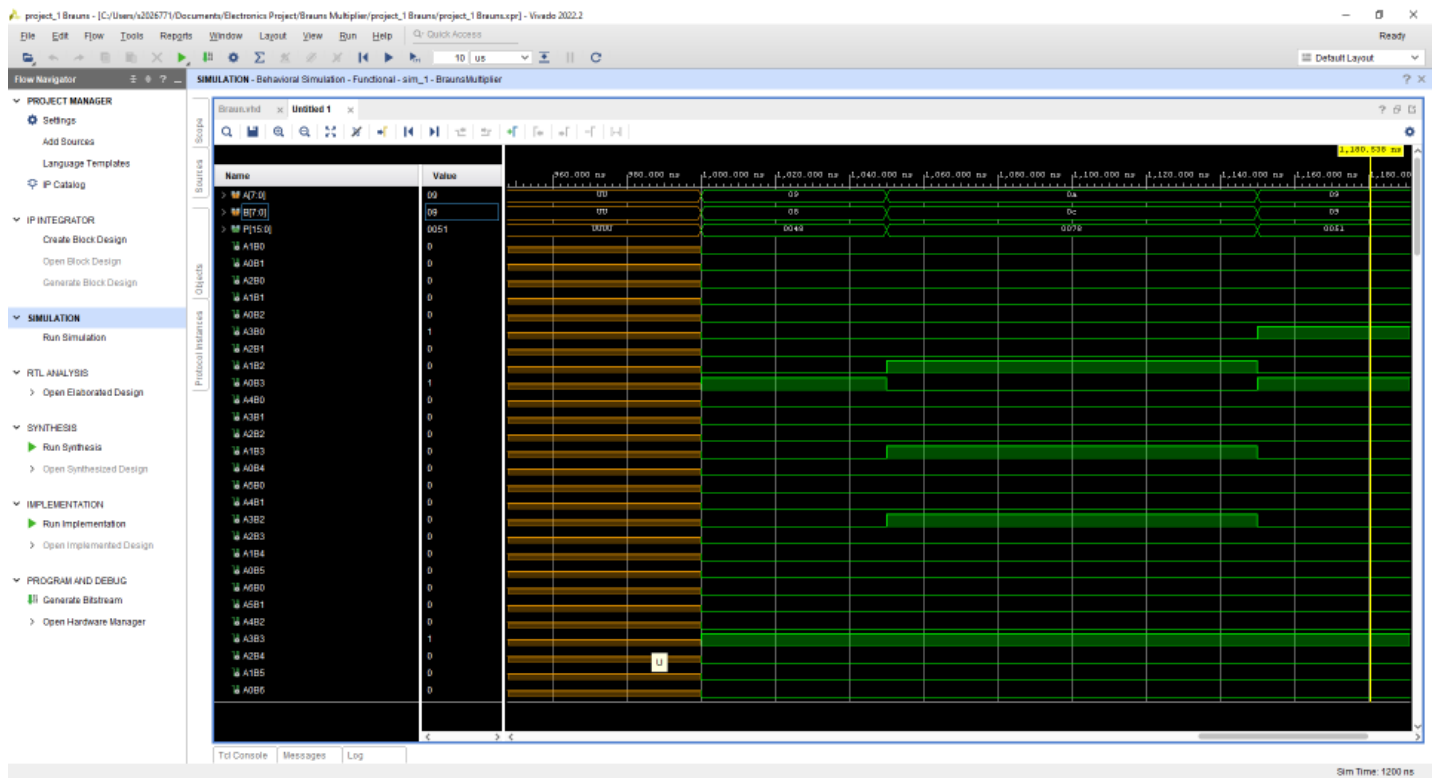


Figure 12

The figure below illustrates the next step, which is to run the synthesis, as shown in the top right-hand corner the synthesis is complete.

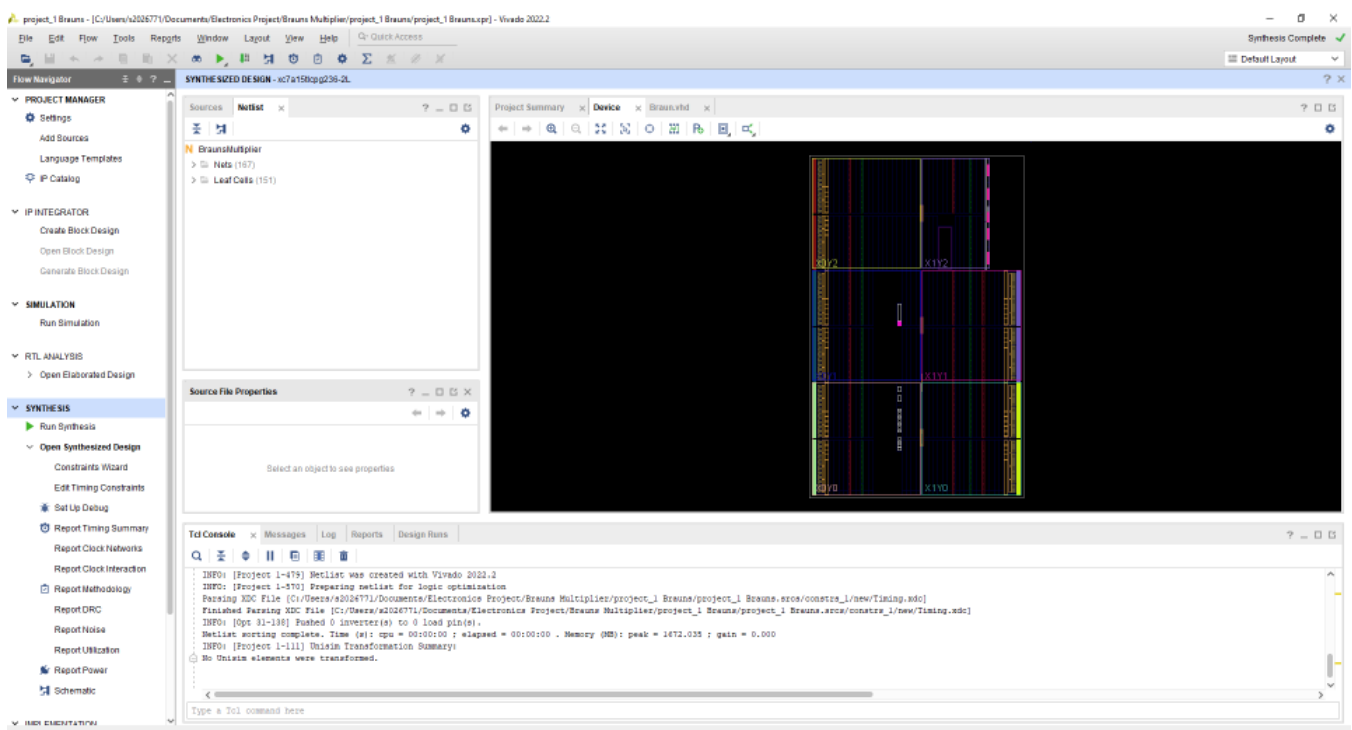


Figure 13

The next image illustrates the schematic diagram.

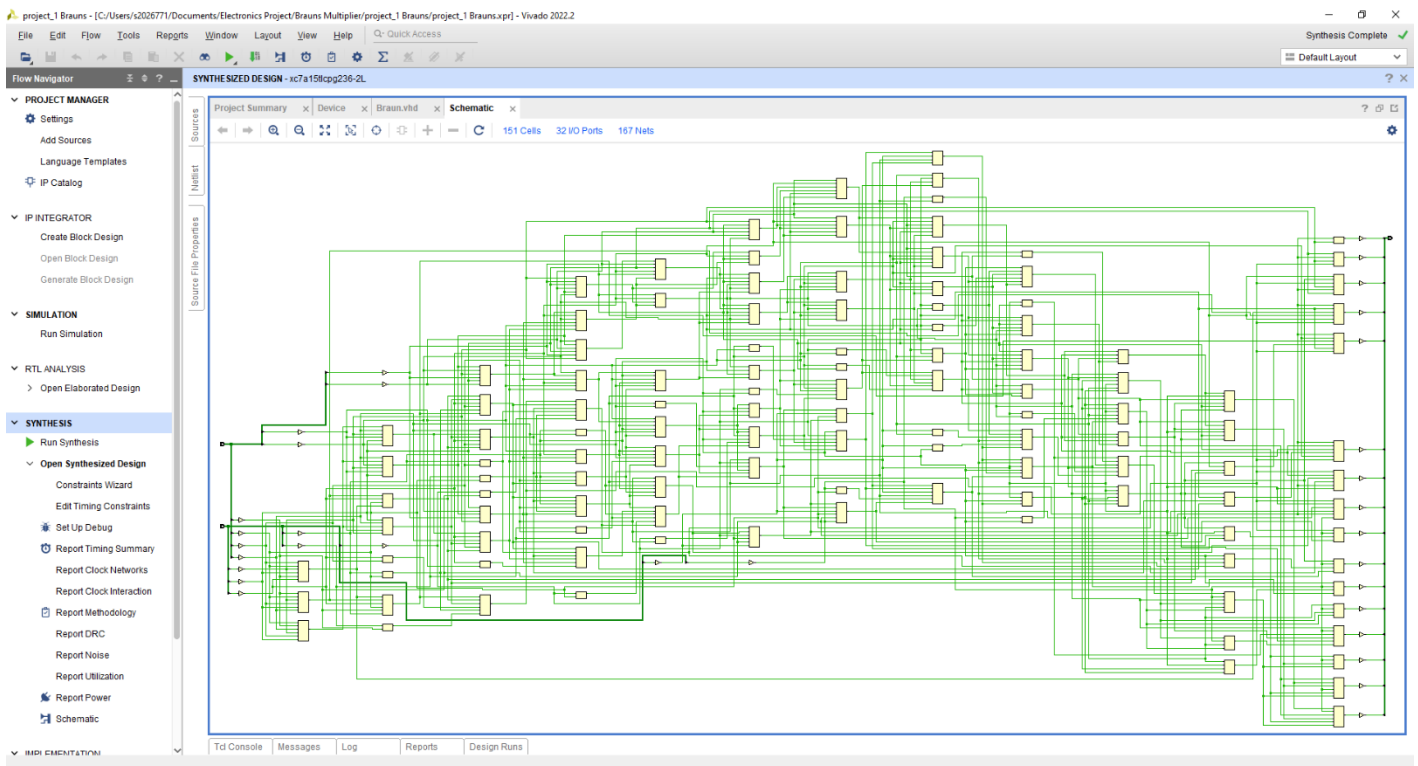


Figure 14

The below illustrates all the I/O ports being assigned to their pins.

Name	Direction	Interface	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vtref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM	Partition Pin Location
A[7]	IN			A4	✓	216									NA
A[6]	IN			A5	✓	216									NA
A[5]	IN			A14	✓	16	default (LVCMOS18)	1.800				NONE	✓ NONE	✓	NA
A[4]	IN			A15	✓	16	default (LVCMOS18)	1.800				NONE	✓ NONE	✓	NA
A[3]	IN			A16	✓	16	default (LVCMOS18)	1.800				NONE	✓ NONE	✓	NA
A[2]	IN			A17	✓	16	default (LVCMOS18)	1.800				NONE	✓ NONE	✓	NA
A[1]	IN			A18	✓	16	default (LVCMOS18)	1.800				NONE	✓ NONE	✓	NA
A[0]	IN			B4	✓	216									NA
B[7]	IN			B6	✓	216									NA
B[6]	IN			B8	✓	216									NA
B[5]	IN			B10	✓	216									NA
B[4]	IN			B15	✓	16	default (LVCMOS18)	1.800				NONE	✓ NONE	✓	NA
B[3]	IN			B16	✓	16	default (LVCMOS18)	1.800				NONE	✓ NONE	✓	NA
B[2]	IN			B17	✓	16	default (LVCMOS18)	1.800				NONE	✓ NONE	✓	NA
B[1]	IN			B18	✓	16	default (LVCMOS18)	1.800				NONE	✓ NONE	✓	NA
B[0]	IN			C15	✓	16	default (LVCMOS18)	1.800				NONE	✓ NONE	✓	NA
P1[15]	OUT			C16	✓	16	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[14]	OUT			C17	✓	16	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[13]	OUT			D17	✓	14	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[12]	OUT			D18	✓	14	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[11]	OUT			D19	✓	14	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[10]	OUT			E18	✓	14	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[9]	OUT			E19	✓	14	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[8]	OUT			F18	✓	14	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[7]	OUT			G2	✓	35	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[6]	OUT			G3	✓	35	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[5]	OUT			G17	✓	14	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[4]	OUT			G18	✓	14	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[3]	OUT			G19	✓	14	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[2]	OUT			H10	✓	35	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[1]	OUT			H2	✓	35	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA
P1[0]	OUT			H17	✓	14	default (LVCMOS18)	1.800	12	✓	✓	NONE	✓ FP_VTT_50	✓	NA

Figure 15

The next figure demonstrates the implementation being run and complete as shown in the top right-hand corner of the image, the timing report summary is also demonstrated, this shows that the worst negative slack is at infinity, a timing restraint will need to be added to ensure that no signal is missing the timing. The total negative slack is at infinity also.

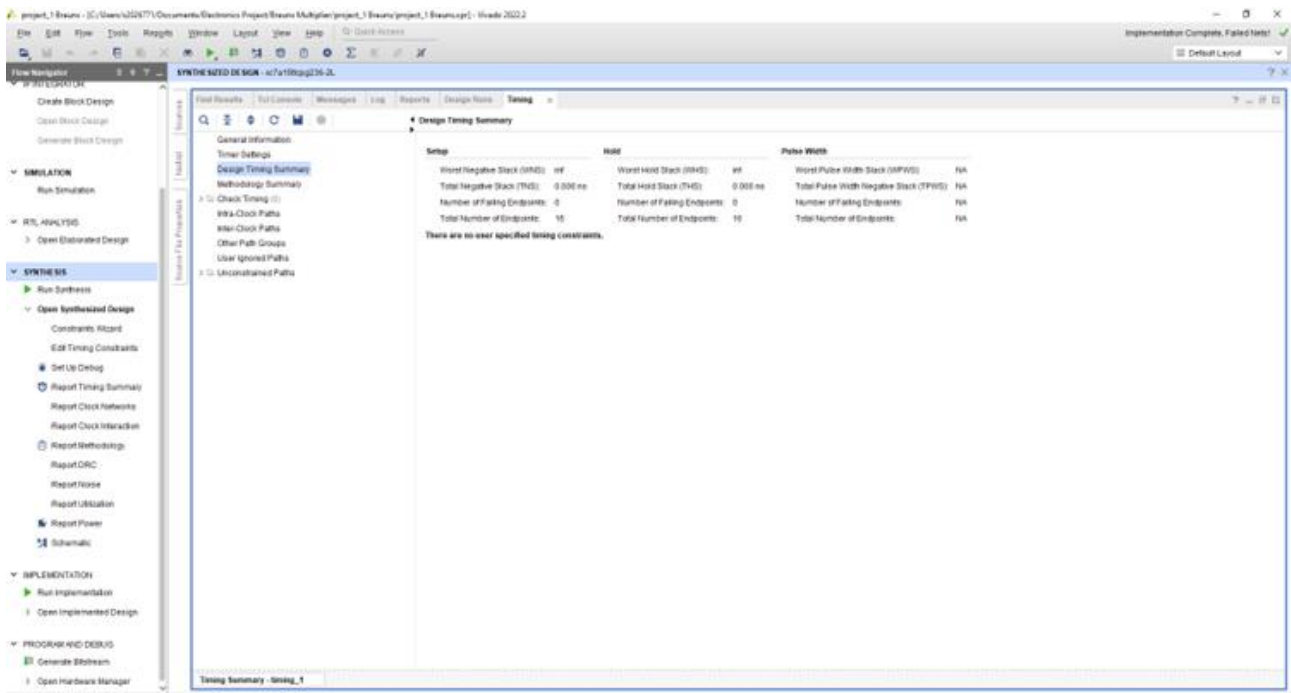


Figure 16

The following figure illustrates the time delays of all the signals.

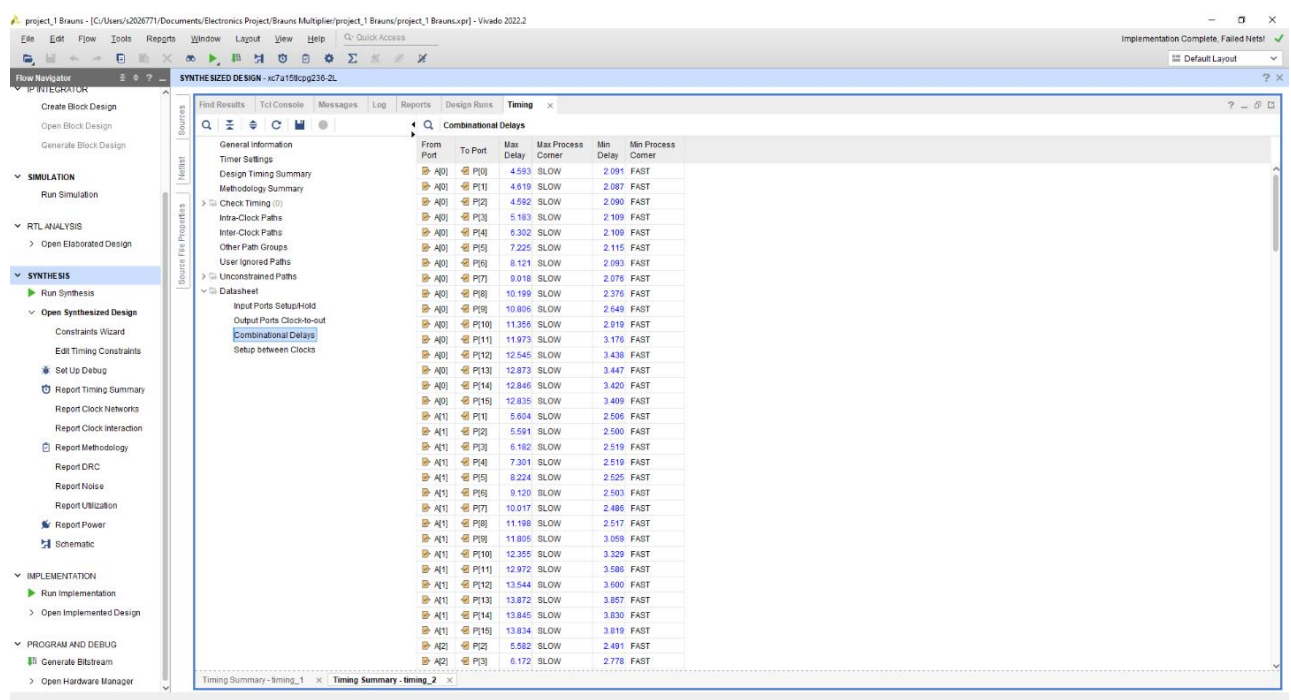


Figure 17

The figure that follows demonstrates the timing constraints, the clock value is set then the longest delay of negative slack is added as a time constraint, this then ensures that no signal is missing the timing.

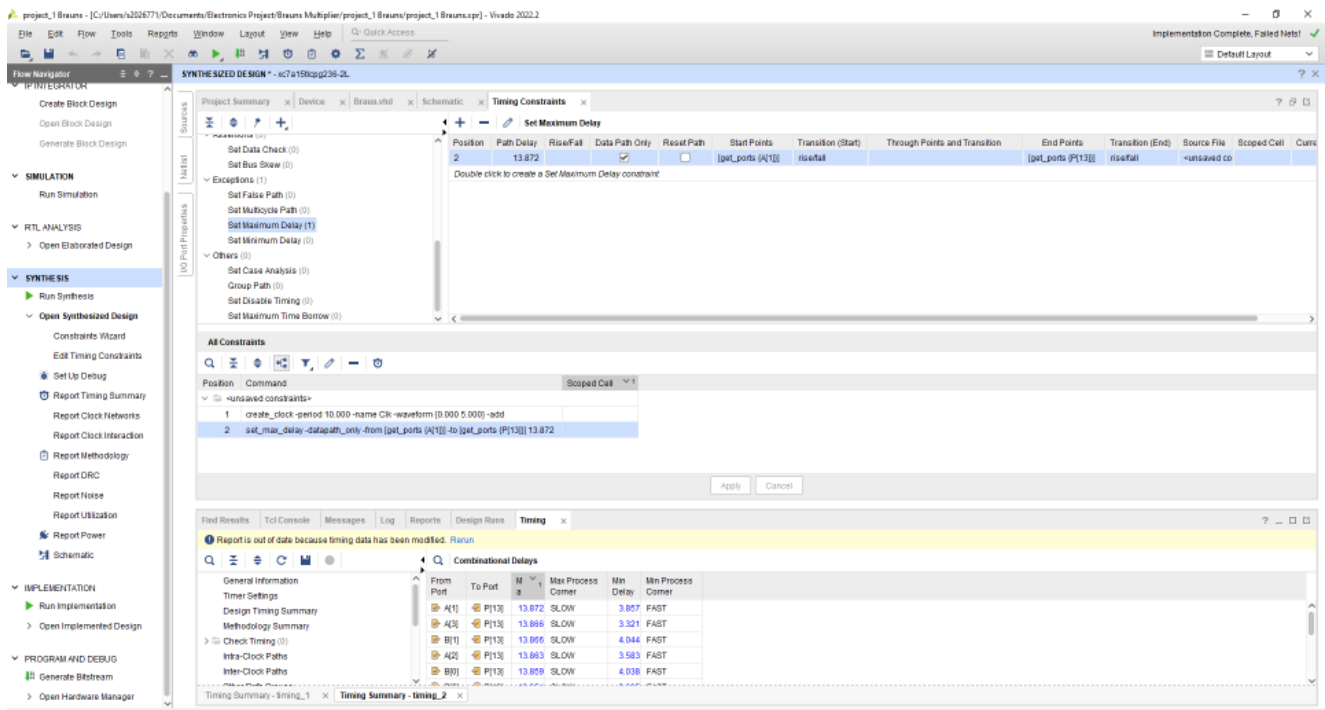


Figure 18

The following figure demonstrates the worst negative slack has been rectified to 0ns, this results in no signal missing the timing.

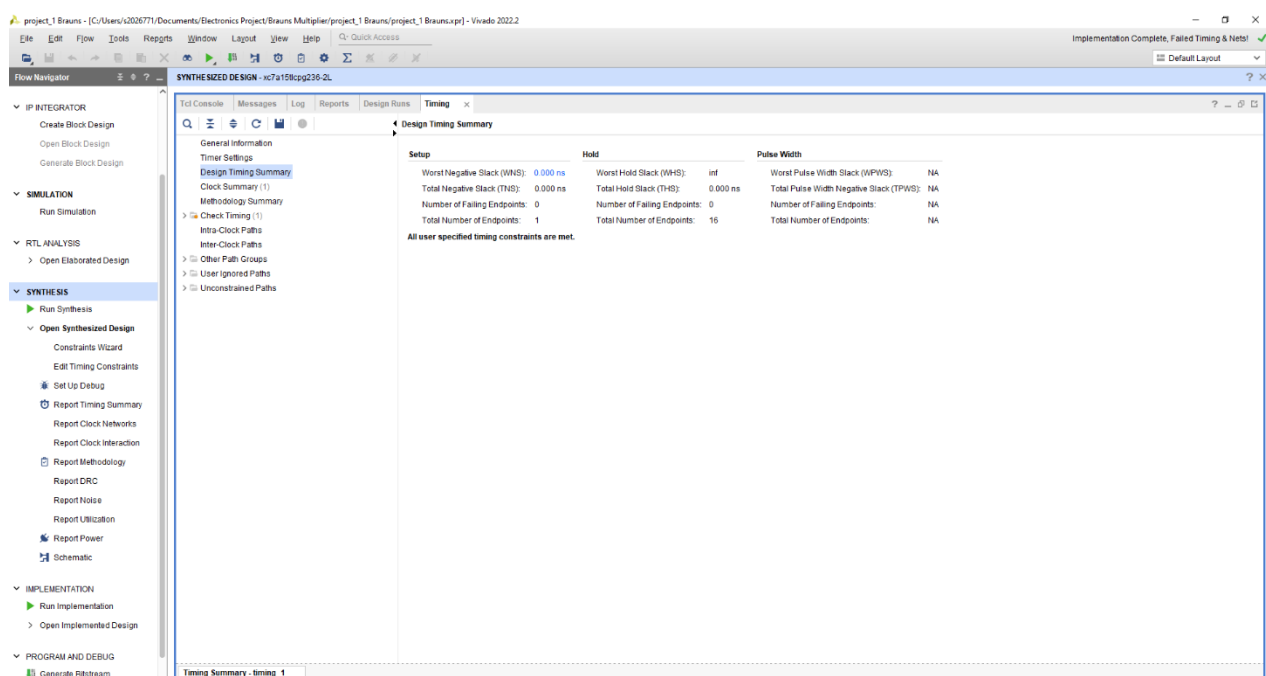


Figure 19

8. Details of testing and results

8.1 Wallace

The following figure demonstrates the power summary of the Wallace design.

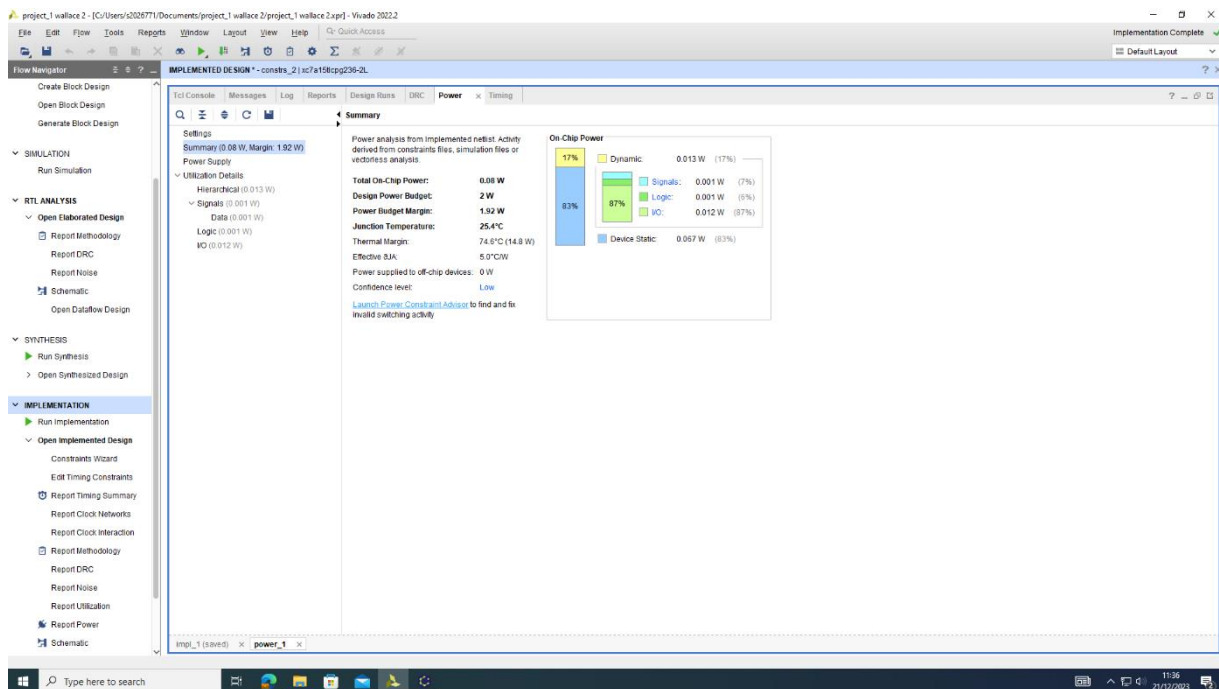


Figure 20

The following figure demonstrates the utilization summary.

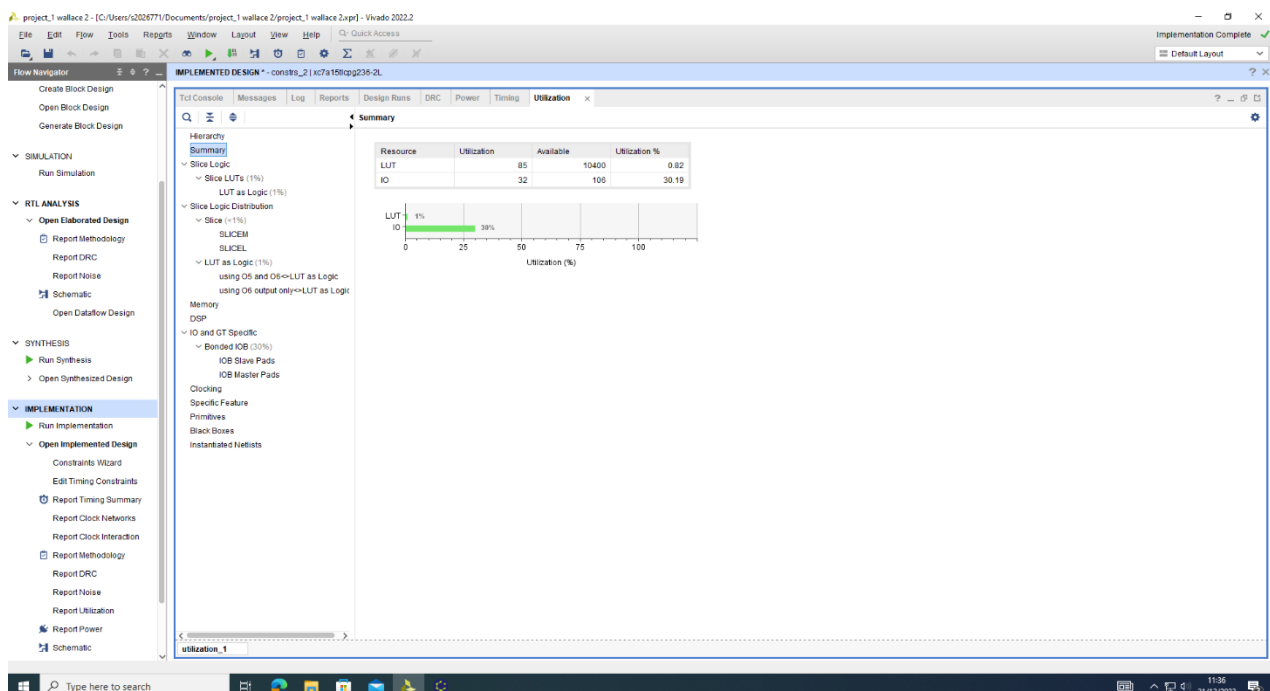


Figure 21

The following figure illustrates the project summary.

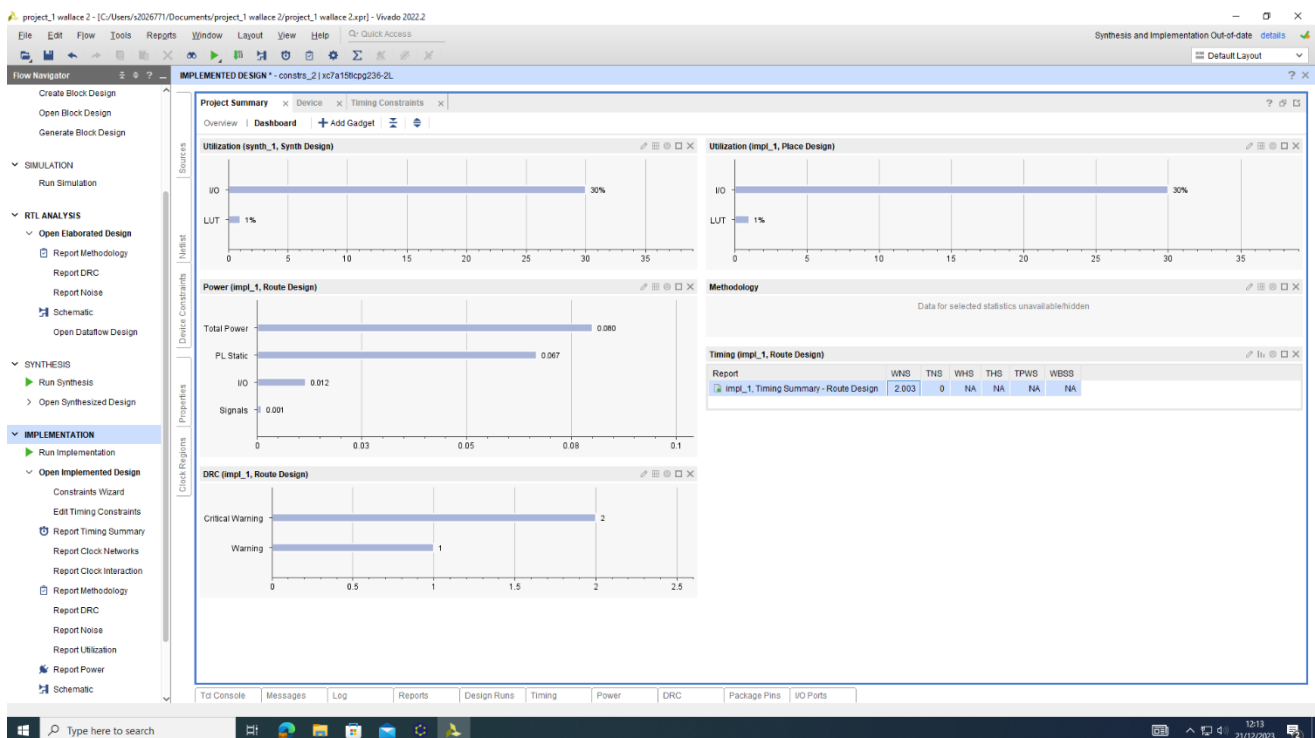


Figure 22

The following figure demonstrates the RTL analysis of the design.

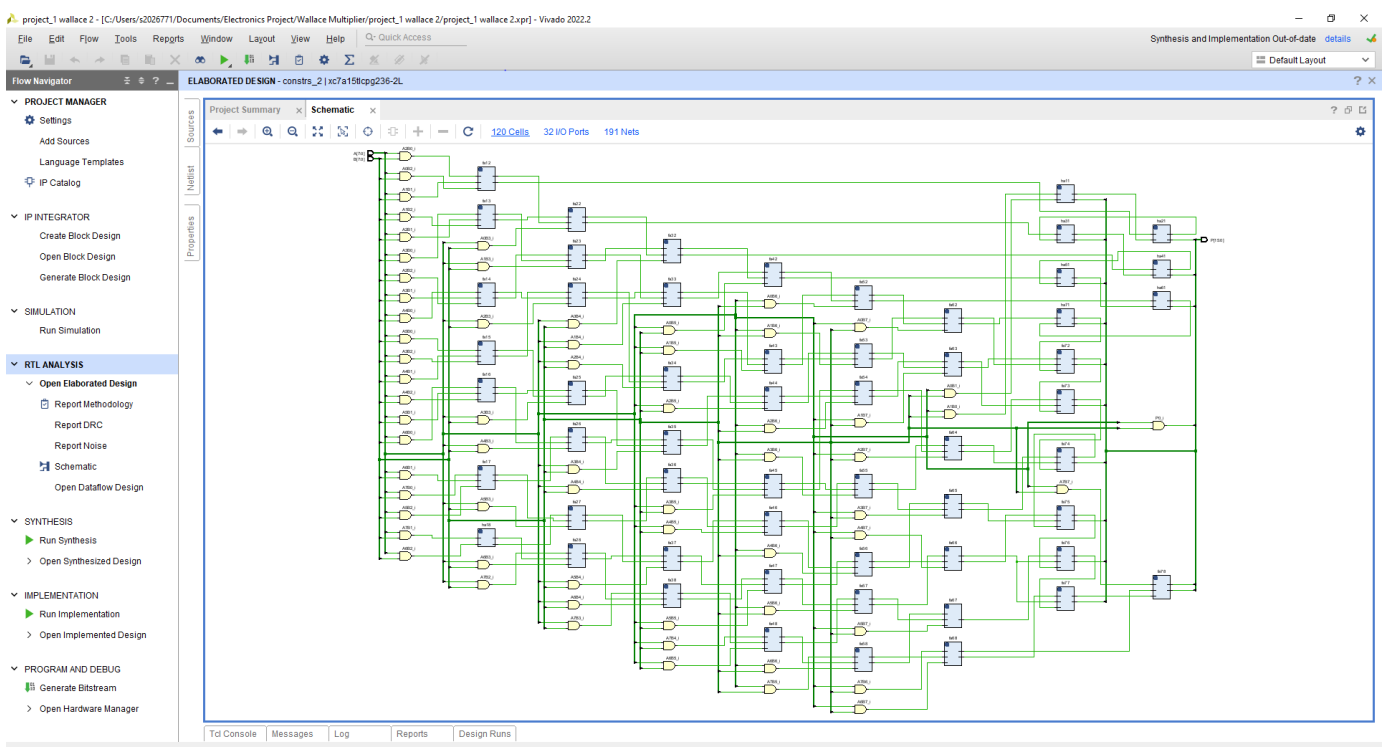


Figure 23

8.2 Brauns

The following figure illustrates the design power budget which is set to 2W.

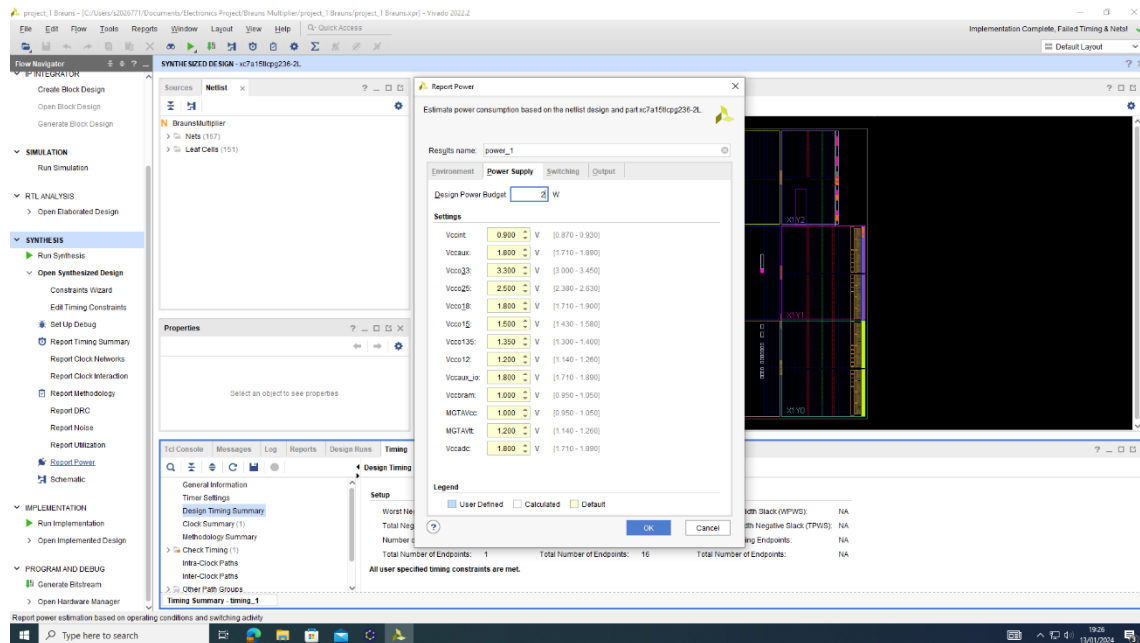


Figure 24

The following figure illustrates the power summary of the Brauns multiplier.

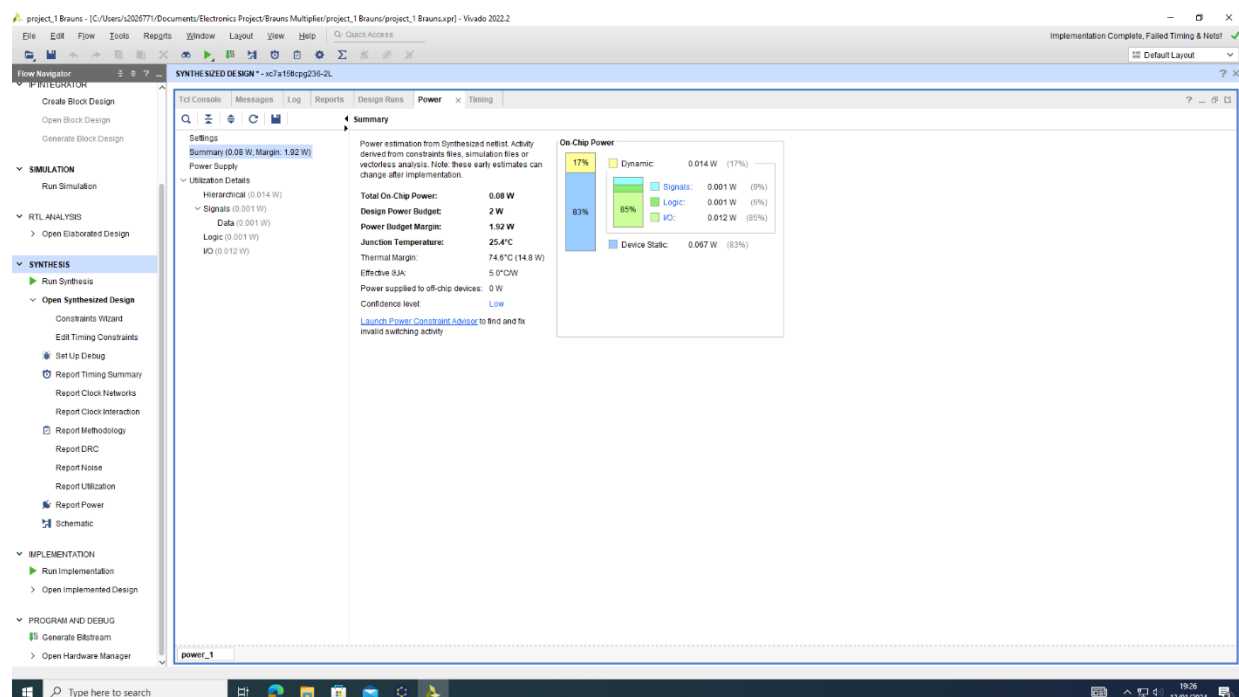


Figure 25

The following figure demonstrates the utilization summary of the Brauns multiplier.

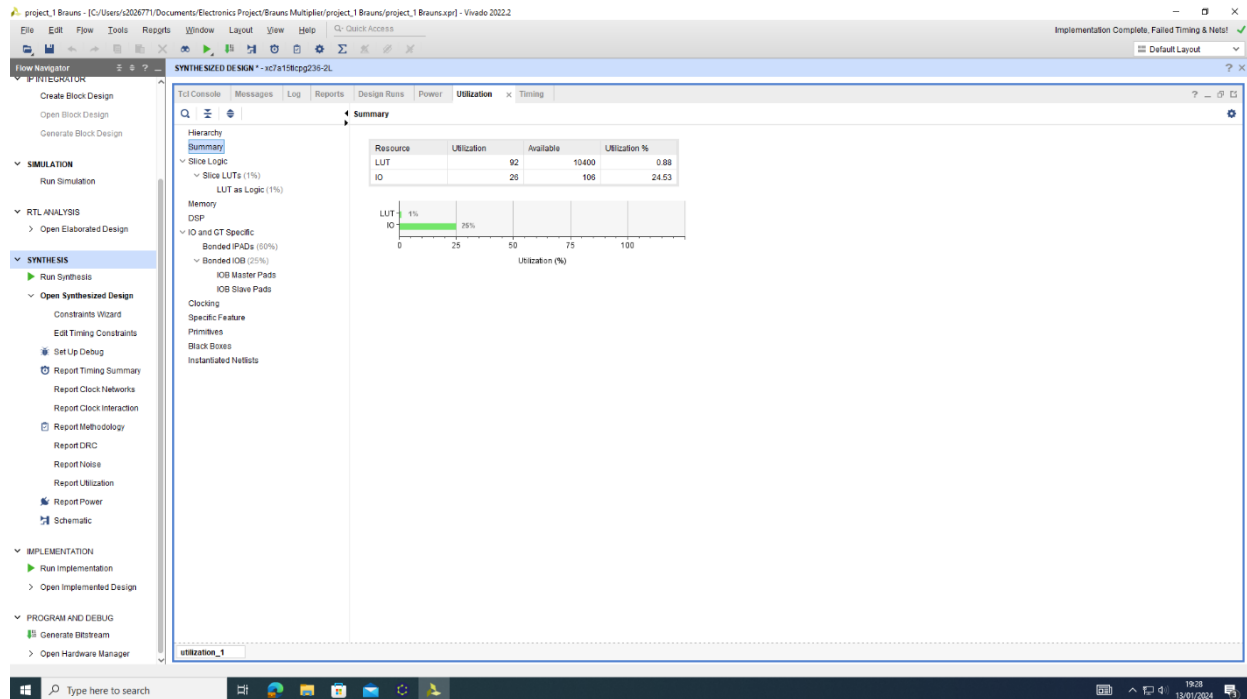


Figure 26

The below figure shows the project summary of the Brauns multiplier.

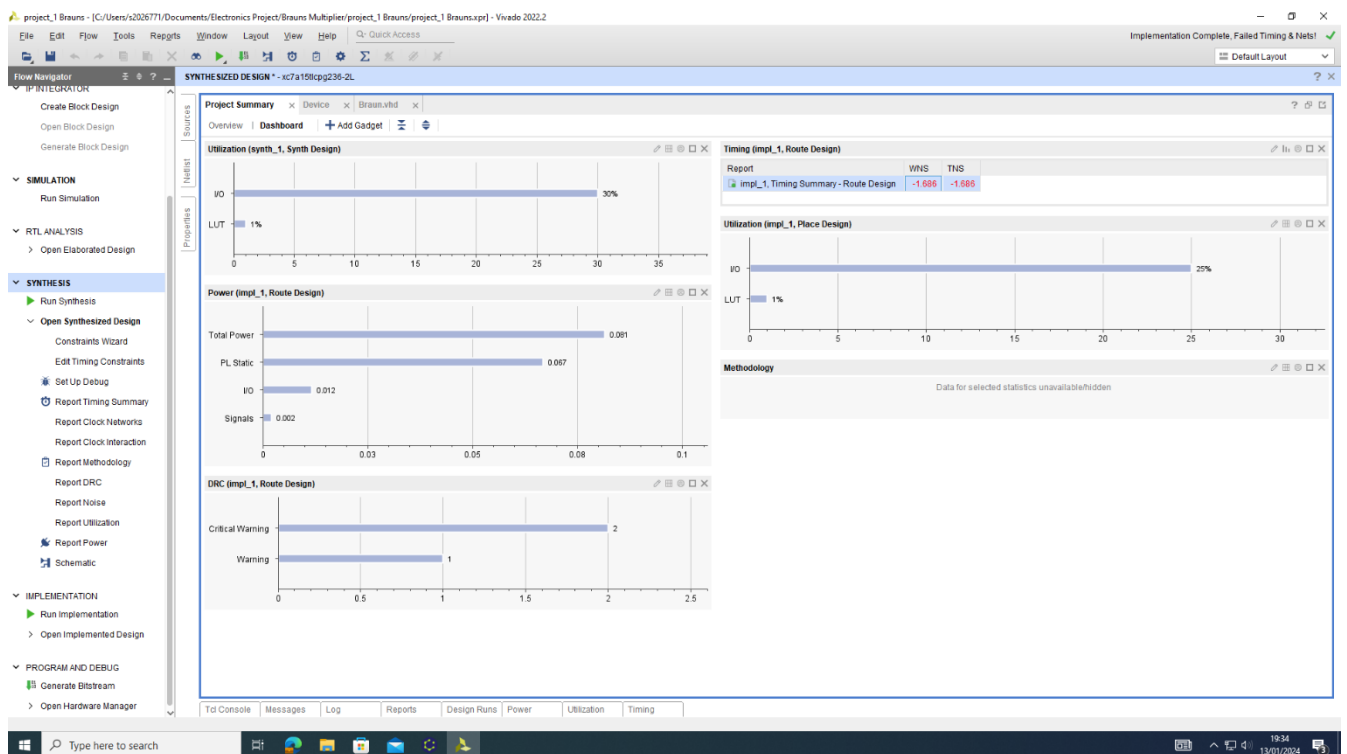


Figure 27

The following figure demonstrates the RTL analysis of the Brauns multiplier.

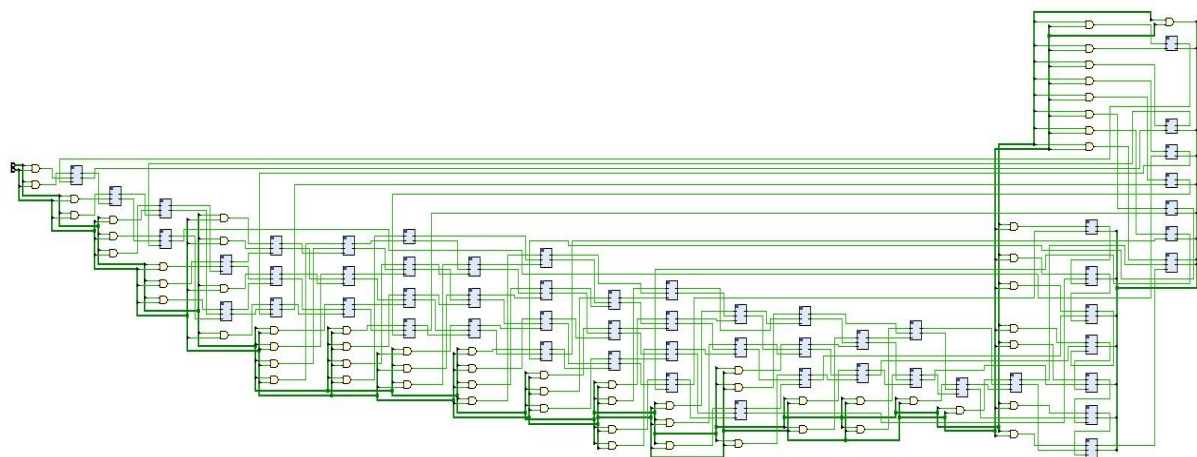


Figure 28

8.3 Longest delay path

8.3.1 Wallace longest delay path

Given that:

- AND Gate Delay = 1 ns
- Full Adder/ Half Adder Delay= 2ns

Longest Delay = Delay of 1st AND Gate + 7 X (Delay in Carry out Half Adder) + 7 X (Delay in Carry Out Full Adder)

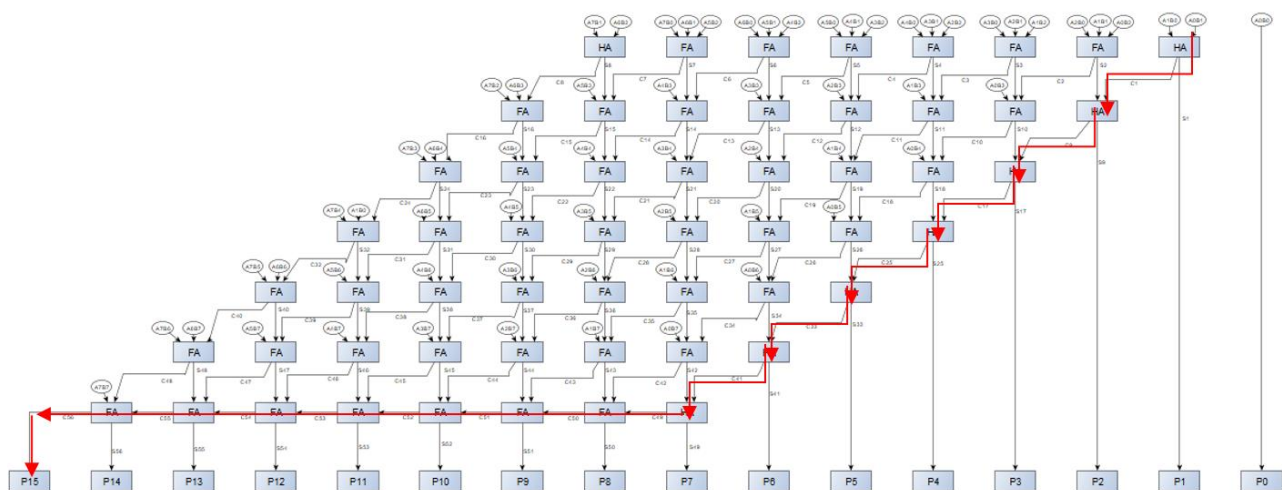


Figure 29

Therefore,

$$\text{Longest Delay} = 1\text{ns} + 7 \times (2\text{ns}) + 7 \times (2\text{ns})$$

$$= 29\text{ns}$$

8.3.1 Brauns longest path delay

Given that:

- AND Gate Delay = 1 ns
- Full Adder/ Half Adder Delay= 2ns

Longest Delay = Delay of 1st AND Gate + 7 X (Delay in Carry out Half Adder) + 13 X (Delay in Carry Out Full Adder)

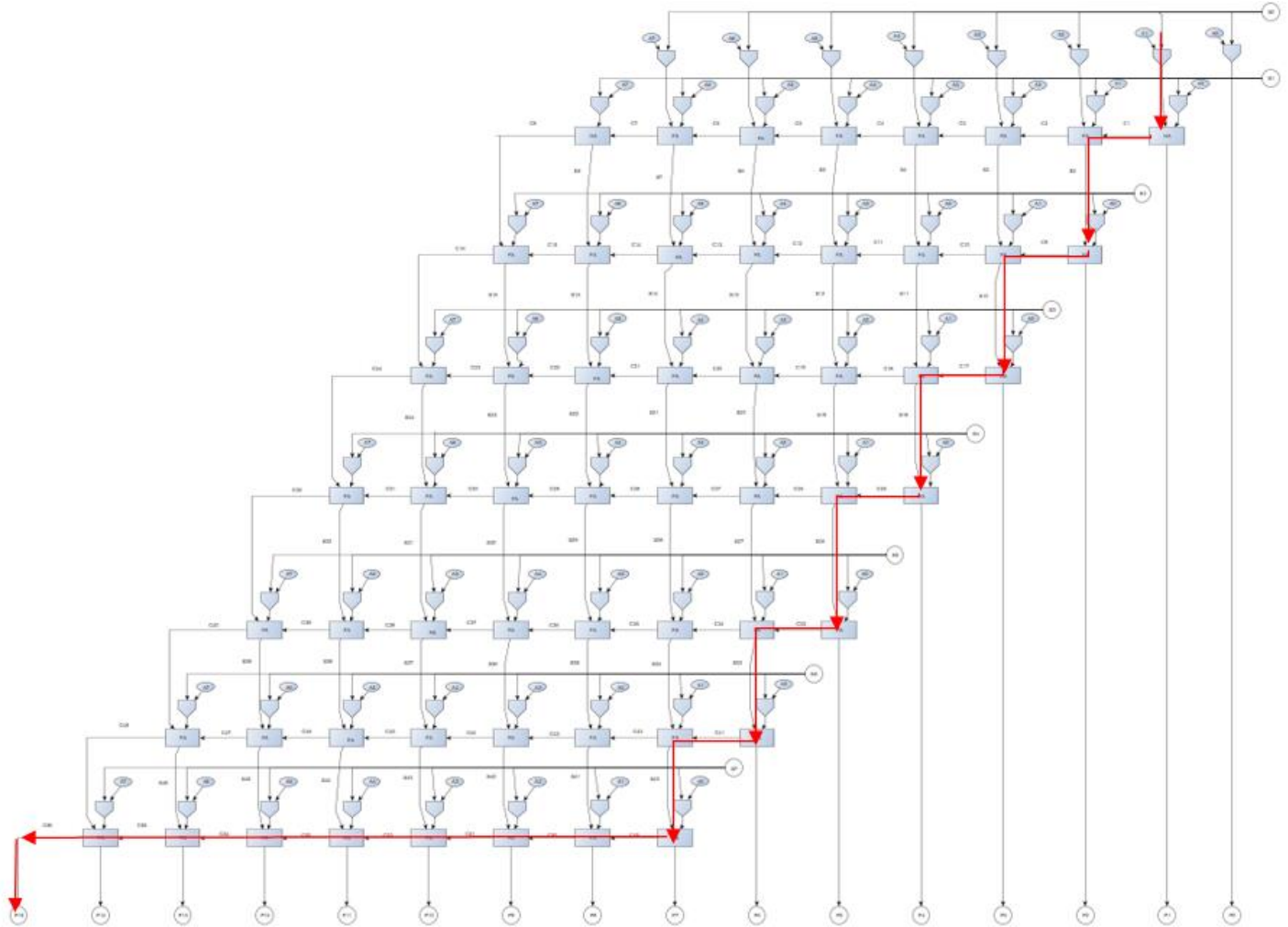


Figure 30

Therefore,

$$\begin{aligned} \text{Longest Delay} &= 1\text{ns} + 7 \times (2\text{ns}) + 13 \times (2\text{ns}) \\ &= 41\text{ns} \end{aligned}$$

9. Conclusion

In conclusion of this report the results have proven that the Wallace multiplier is has the fastest speed for transmitting signals, these results support the research that was conducted at the start of the report. The results for each are listed below:

Wallace multiplier

- Worst negative slack = 2.003ns
- Total negative slack = INF
- Power 0.13 W
- Utilisation = 30.19 %

Brauns Multiplier

- Worst negative slack = -1.686ns
- Total negative slack = -1.686ns
- Power 0.14 W
- Utilisation = 25%

The results show that the Wallace multiplier had signals that were faster than the timing criteria, the Brauns multiplier proves that the signals are slower than the timing criteria, both will use a higher power consumption, the Wallace uses 0.01W less power than the Brauns Multiplier.

Braun's multipliers and Wallace multipliers are both efficient techniques used in digital circuit design for multiplication operations. Braun's multipliers offer a straightforward implementation and is suitable for small to medium-sized multiplication tasks. It provides a balanced trade-off between speed and area utilization. On the other hand, Wallace multiplier excels in larger multiplication tasks, as it significantly reduces the number of partial products and minimizes the overall circuit delay.

Both techniques have their advantages and limitations. Braun's multiplier is simpler to implement and requires fewer resources, making it suitable for low-power and area-constrained applications. However, it may suffer from longer delays due to the nature of its operation. Wallace multiplier, on the other hand, offers faster multiplication speeds but requires more complex circuitry.

In summary, the choice between Braun's multiplier and Wallace multiplier depends on the specific requirements of the application. Braun's multiplier is a good choice for smaller multiplication tasks with limited resources, while Wallace multiplier is more suitable for larger multiplication tasks that prioritize speed and performance.

Bibliography

- [H. Chugh, "Research Gate," [Online]. Available: https://www.researchgate.net/profile/Hemanshi-Chugh/publication/365668555_Designing_of_Various_types_of_multipliers_and_their_implementation_on_FPGA/links/637dc2422f4bca7fd08303ef/Designing-of-Various-types-of-multipliers-and-their-implementation-on-FPGA..
- [S. P., "IEEE Comparison of Braun Multiplier and Wallace Multiplier Techniques in VLSI," [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8605173>.
- [K. G. S. T. S. Himanshu Bansal, "Wallace Tree Multiplier Designs: A Performance Comparison," IISTE, 2014.
- [S. Mathiassen, "Power optimized multipliers," Norwegian University of Science and Technology, 2010.
- [C. F. S. C. U. Roger Lipsett, VHDL: Hardware description and design, 2012.
- [IEEE, 1076-2008 - IEEE Standard VHDL Language Reference Manual, IEEE, 2009.
- [M. H. A. M. a. M. N. Muhammad H. Rais, "FPGA Realization of Braun's Multipliers," IMECS, Hong Kong, 2012.