

Comparison of Brauns Multiplier and Wallace tree multiplier using VHDL

by Oliver Bunyan

Submission date: 16-May-2023 03:26PM (UTC+0100)

Submission ID: 206464240

File name: 34448_Oliver_Bunyan_Comparison_of_Brauns_Multiplier_and_Wallace_tree_multiplier_using_VHDL_497727_1372109090.docx (4.92M)

Word count: 5397

Character count: 28907

ACDA6001 – Electronics 2

DESIGN AND COMPARISON OF A BRAUNS MULTIPLIER AND WALLACETREE MULTIPLIER USING VHDL

Bunyan, Oliver
UWTSD SWANSEA
C&P Engineering Services Ltd

[Table of contents](#)

Table of contents	0
1. Introduction.....	2
2. Background research	4
2.1 Field Programmable Gate Array	4
2.2 Multipliers	4
3. Information and analysis of similar products	5
3.1 Booths Multiplier	5
4. Circuit Design.....	6
4.1 Braun's Multiplier	10
4.2 Wallace tree multiplier	12
5. A flowchart/pseudocode showing the system operation.....	14
6. FPGA Diagram	15
6.1 8x8 Braun's Multiplier FPGA	15
6.2 Wallace tree Multiplier FPGA Diagram	20
7.1 Braun's Multiplier	25
7.1.1 Schematic Diagram.....	25
7.1.2 Timing of Braun's Multiplier.....	28
7.2 Wallace Tree Multiplier	30
7.2.1 Schematic Diagram	30
7.2.2 Timings of Wallace Tree Multiplier	33
8. Simulation Results	35
8.1 Braun's Multiplier	35
8.2 Wallace Tree	45
9.1 8x8 Braun's multiplier and 8x8 Wallace tree multiplier results.....	56
12. Appendices	59
12.1 Appendix 1	59
12.2 Appendix 2.....	64
12.2.1 Braun's Multiplier VHDL Code	64
12.2.2 Braun's Multiplier Test Bench	69
13. References	71

1. Introduction

This report contains the design of two different types of multipliers using the language VHDL. Multipliers are used to carry out multiplication between two binary numbers. Multipliers are used widely in digital signalling processing (DSP) as well as other applications. Within DSP these are used for filtering, With the increase in equipment requiring high-speed processing, multipliers have become important and are contained within over 70% of microprocessors. Each different multiplier has different factors that will make one of the multipliers work  2 than the other, this would be with the speed times of each of the multipliers, the Accuracy, area the multiplier covers such as numbers of LUTs (Look up tables) and the power consumption. [1]

 3

The report aims to analyse the results of two given multiplier types. Each of the multipliers will be written in VHDL code into a software called Vivado. This software allows the multipliers to be simulated and synthesised onto an FPGA. After analysis, a comparison can be made on the results given from the simulation builds. The two multipliers that were designed are an 8x8 Braun's Multiplier and an 8x8 Wallace tree multiplier. Firstly, a logic circuit was designed following  4 parallel multiplication that aided with understanding what gates were needed for the build. A code was then written for both multipliers using VHDL code within the Software Xilinx Vivado. Both multipliers were  5 analysed by comparing the power consumption of each multiplier, the number of gates each multiplier used and their timings. The same delays were added to both designs to allow accurate comparisons to be made. There are other multipliers out there that have different methods of multiplication which have been talked about within the report.

2. Background research

VHDL code is a language that is Hardware description language (VHDL) and is used within integrated circuits (ICs). Using VHDL combined within an IC allows for very high speeds of dataflow. VHDL Coding is used to describe the function of an electronic circuit as done within this report, which will be implemented onto an FPGA (Field Programmable Gate Array). When ICs were invented, they allowed the use of digital circuits to fully progress, this is due to the size of the circuits shrinking massively and the complexity of the IC circuit containing thousands of transistors and being able to carry out complex functions. [2] Binary multipliers are done by simply calculating partial products of the m and n bits.

2.1 Field Programmable Gate Array

FPGA is a programmable logic device. They are a type of IC and they can be configured to be able to be any electronic circuit that the user wants it to be. They can be configured and unconfigured as many times as it likes as it uses a RAM memory type (Random access memory). When using an FPGA you are creating hardware, this can be done using languages such as VHDL coding.

2.2 Multipliers

Multipliers are widely used in the electronic industry and play a large role in processing digital signals. Multipliers are when multiplication and addition of Binary numbers are required, systems that require these are digital signalling processes as well as a microprocessor. Speed is important when it comes to processing therefore the goal of the multiplier is to be as fast as possible and maintain a low power efficiency. A large 70% of the processes within a microprocessor require the use of multiplication and addition due to the increase of applications requiring signal processing. The goal when designing a multiplier is to have the least amount of chips available therefore impact consumption unit that has a high speed and a power outage that is low. [3]

3. Information and analysis of similar products

There are numerous different multipliers available to be used to carry information through Integrated circuits or FPGA's. Multipliers can come in two different types, these being either a serial multiplier or a parallel multiplier. Different multipliers have different benefits to different circuits as well as pros and cons.

3.1 Booths Multiplier

Booth's multiplier is another widely used multiplier to transmit signals. Booth's multipliers multiply two signed numbers in the -2 's complement. They are very efficient and speed up the multiplication by using a flowchart system. In a study carried out the booth recorded Wallace tree was said to have the fastest response with that being 67% faster than a Wallace tree. When there is a 00 or 11 there is no arithmetic operation and the values only shift. When there is a 01 the multiplicand is added to the left half of the product and its shifted right. When there is a 10 the multiplicand is subtracted from the left half of the product and shifted. The flowchart is shown below in figure 1. [4]

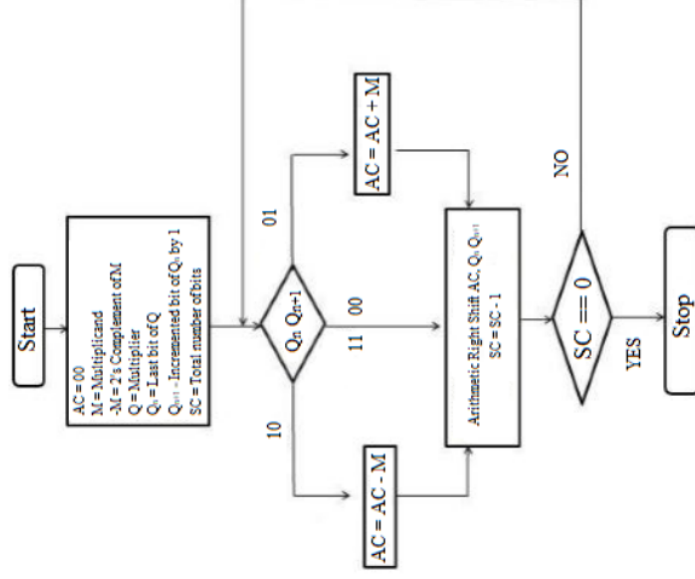
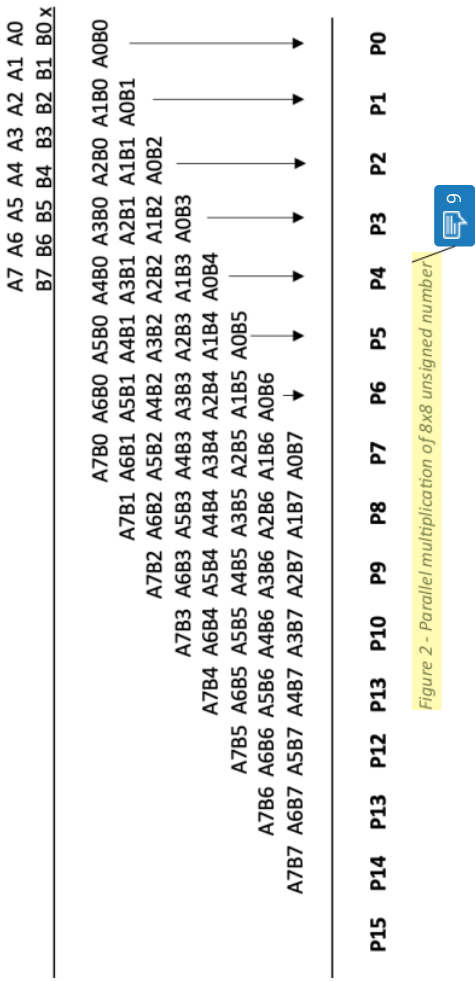


Figure 1 - Booth's algorithm Flowchart [7]

4. Circuit Design

The Braun's and Wallace tree multipliers were designed to be used to compare the results. Parallel multiplication was used for Braun's multipliers to start the design and to create the logic circuit design for the multiplier. With the use of this, a VHDL code was written out within Xilinx Vivado. Figure 2 shows the designed parallel multiplication used, achieving a 16-bit output from 2 8-bit inputs.



Prior to the multipliers being designed using the VHDL code, research was carried out to find a chip that was suitable to be used for this multiplier. It required to have at least 16 inputs and 16 outputs to accommodate the inputs and outputs of the multiplier. The chip to be used is the Nexys a7 100T – XC7A100T-1CSG324C. Using this chip, the pinout was found from the reference manual which allowed the correct inputs and output pins to be implemented within the Xilinx software on the FPGA. The pinout is displayed in figure 3. This chip and chip pinout configuration was used for Braun’s multiplier and the Wallace tree multiplier.

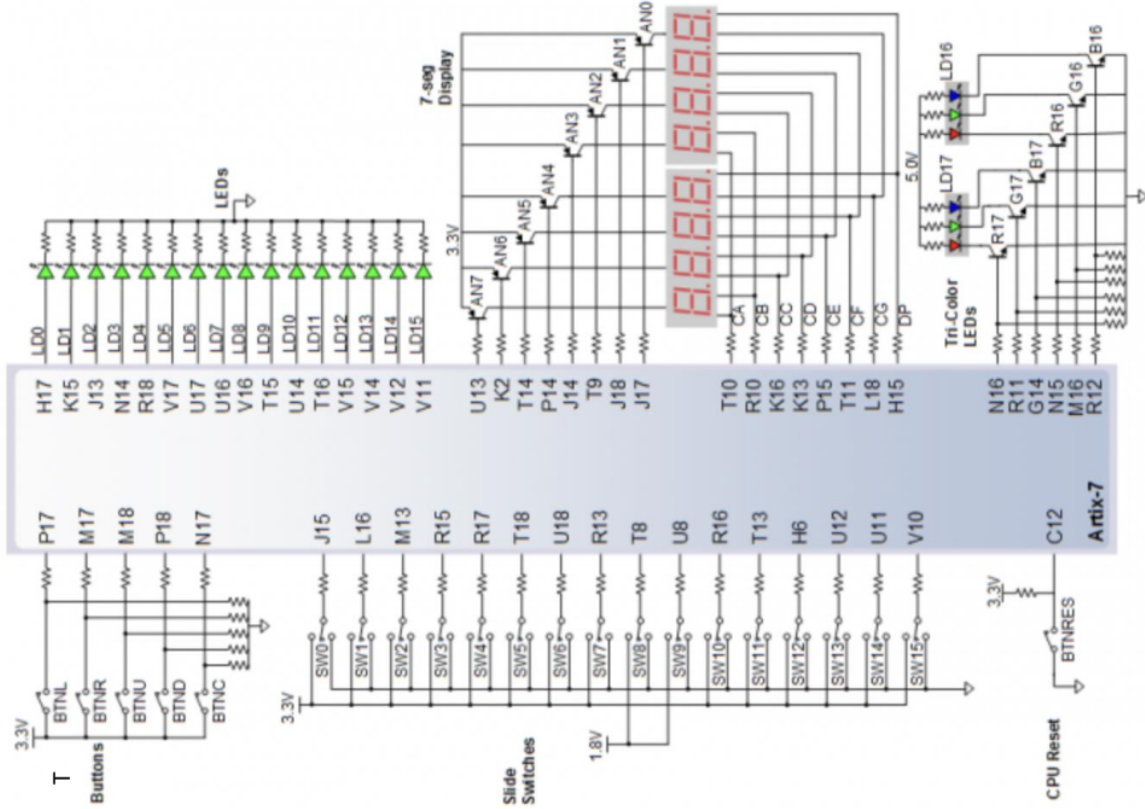


Figure 3 - Nexys A7 100T-1CSG324C Pinout [2]



Find Results															
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			J15	☒	15	default (LVCMOS18)	1.800				NONE	NONE		N/A
A[6]	IN			L16	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
A[5]	IN			M13	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
A[4]	IN			R15	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
A[3]	IN			R17	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
A[2]	IN			T18	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
I/O Ports in 'Schematic' (32)															

Figure 4 - Inputs A7-A2 mapped onto a chip on Vivado using Nexys A7 100T pinout

Find Results															
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[1]	IN			U18	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
A[0]	IN			R13	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
B[7]	IN			T8	☒	34	default (LVCMOS18)	1.800				NONE	NONE		N/A
B[6]	IN			U8	☒	34	default (LVCMOS18)	1.800				NONE	NONE		N/A
B[5]	IN			R16	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
B[4]	IN			T13	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
I/O Ports in 'Schematic' (32)															

Figure 5 - Inputs A1-A0 and -B7-B4 mapped onto a chip on Vivado used Nexys A7 100T pinout

Find Results															
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
B[3]	IN			H6	☒	35	default (LVCMOS18)	1.800				NONE	NONE		N/A
B[2]	IN			U12	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
B[1]	IN			U11	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
B[0]	IN			V10	☒	14	default (LVCMOS18)	1.800				NONE	NONE		N/A
P[15]	OUT			H17	☒	15	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50		N/A
P[14]	OUT			K15	☒	15	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50		N/A
P[13]	OUT			J13	☒	15	default (LVCMOS18)	1.800	12			NONE	FP_VTT_50		N/A
I/O Ports in 'Schematic' (32)															

Figure 6 - Inputs B3-B0 and Outputs P15 to P13 mapped onto a chip using Nexys A7 100T pinout

Tcl Console Messages Log Reports Design Runs Find Results x DRC Power Timing																
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	
P[12]	OUT			N14	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[11]	OUT			R18	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[10]	OUT			V17	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[9]	OUT			U17	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[8]	OUT			U16	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[7]	OUT			V16	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[6]	OUT			T15	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
I/O Ports in 'Schematic' (32)																

Figure 7 - Outputs P12-P6 mapped onto a chip using Nexys A7 100T pinout

Tcl Console Messages Log Reports Design Runs Find Results x DRC Power Timing																
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	
P[6]	OUT			T15	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[5]	OUT			U14	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[4]	OUT			T16	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[3]	OUT			V15	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[2]	OUT			V14	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[1]	OUT			V12	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
P[0]	OUT			V11	☒	14	default (LVCMOS18)	1.800		12		NONE	FP_VTT_50		N/A	
I/O Ports in 'Schematic' (32)																

Figure 8 - Outputs P6-P0 mapped onto a chip using the Nexys A7 100T pinout

Within figures 4 to 8 are the I/O ports having been mapped with the use of the pinout of the chip In use. All the inputs have used dedicated input pins and the outputs have been mapped to dedicated output pins. The same input and outputs were mapped to the same pin numbers on each of the two different multipliers. The process above of mapping the I/O's to dedicated pins was done after the VHDL code was written and simulated for the test results.

4.1 Braun's Multiplier

Using the 8x8 parallel multiplication in figure 2 a logic circuit was designed of an 8x8 bit Braun's multiplier. The Braun's multiplier in figure 9 consists of 64 AND gates, 48 Full adders and 8 Half adders. The AND gates are used to combine each individual input bit A0-A7 and B0-B7 together for multiplication. The full adders are used to take 3 inputs and give out 2 outputs, a sum, and a carry. The half-adders take 2 inputs and give out 2 outputs, a sum, and a carry.

With the use of the logic circuit diagram in figure 9 a VHDL code was written. Where each Sum output and carry output was given a number which aided in the design of the code. The code was written in structural type .



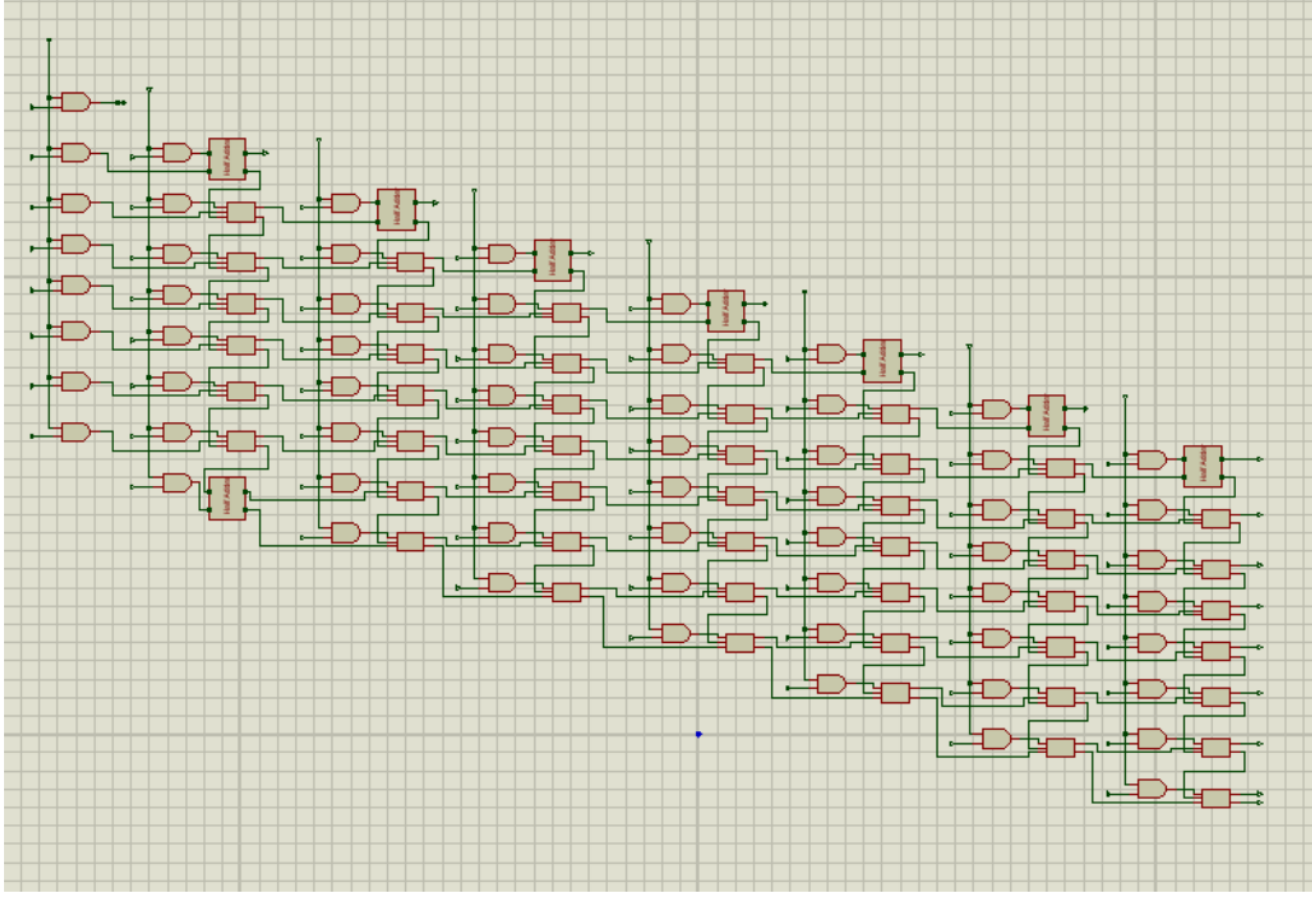


Figure 9 – 8x8 Braun's Multiplier logic circuit diagram

12

4.2 Wallace tree multiplier

After Braun's multiplier's VHDL code was written and run successfully. The next task was to write the VHDL code for the Wallace tree multiplier. With the use of the parallel multiplier in Figure 2 and the dot diagram in Figure 10 a logic circuit diagram could be built. With the use of all these steps a VHDL code in structural language was written, implementing all the components required for the multiplier, before then port mapping the inputs and outputs.

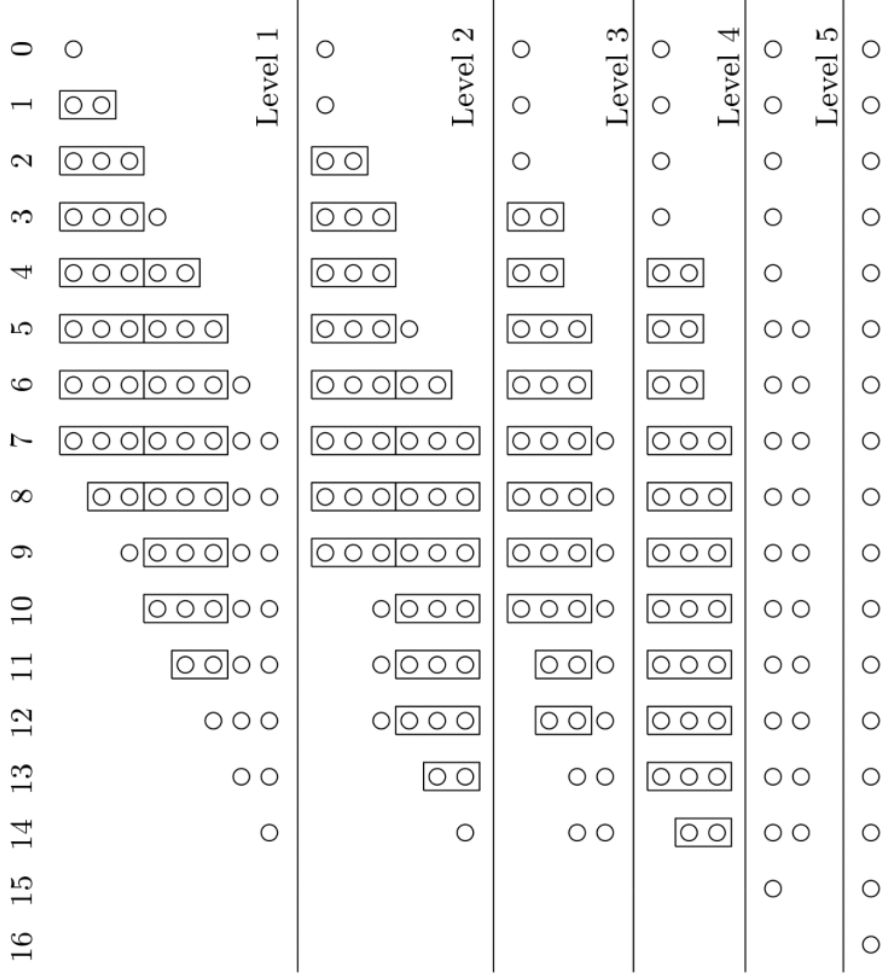


Figure 10 - 8x8 Wallace tree multiplier dot diagram [1]



The use of the dot diagram made the writing of the VHDL code easier to write and understand. Every 2 dots within a rectangle represent a Half adder and every 3 dots in a rectangle represent a full adder. When a dot is on its own without being closed off by a rectangle it gets carried on to the next level, where it can be adjoined to sums or carry from the outputs of the level before them. Each sum of the Half adder or full adder in the level above goes into the column directly below in the next level down. And each carry in the level above goes into column one to the left directly below. If there are two adders directly above and two adders directly above and to the right, this gives 4 partial products into the

level below, 2 sums and 2 carries. In any order 3 of the 4 partial products can be placed into a full adder, with the remaining partial product carried through to the level below again.

Using this method allowed the code to be written. The last line of the dot diagram represents the partial products P0-P15.



By the use of figure 10 it gave the circuit 48 Full adders and 16 half adders. Both the multipliers had 64 AND gates, this was to combine the inputs A0-A7 and B0-B7, which is 8 multiplied with 8 = 64. The reason there are not 47 full adders visible in figure 10 is due to the 2 dots visible in level 5 become full adders due to the carry of the adders to the right of them. The first two dots stay a half adder, then the carry from that half adder gets taken to the half adder to the left of them, making them a full adder, this links all of them through like an adder array, and the last individual dot will become a half adder from the carry to the right. This representation is displayed in the logic circuit in figure 11.

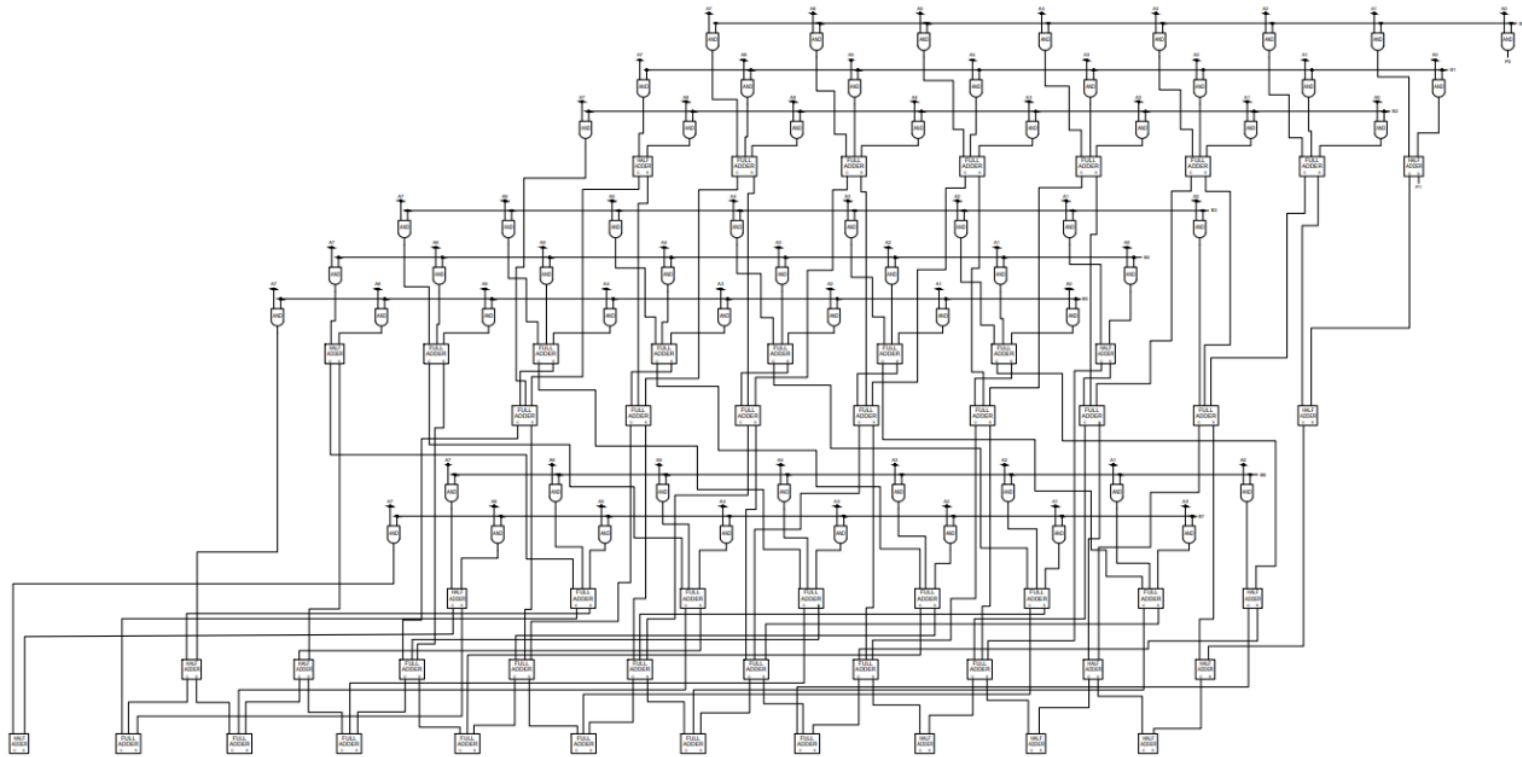


Figure 11 - Wallace tree multiplier logic circuit to Stage 3

15

5. A flowchart/pseudocode showing the system operation

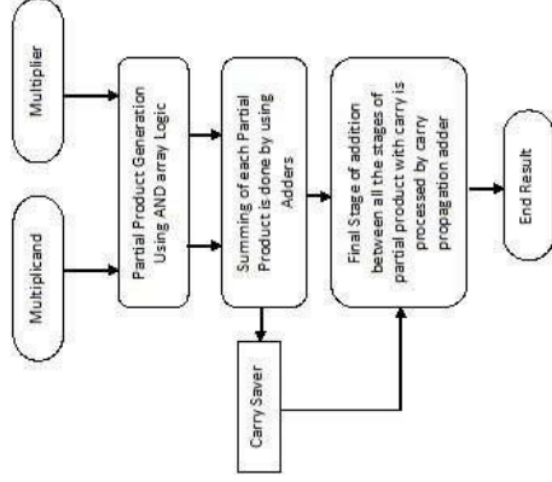


Figure 12 - Flowchart of Wallace Tree Multiplier

Figure 13 is a flowchart of the process of a Wallace tree multiplier.

6. FPGA Diagram

6.1 8x8 Braun's Multiplier FPGA

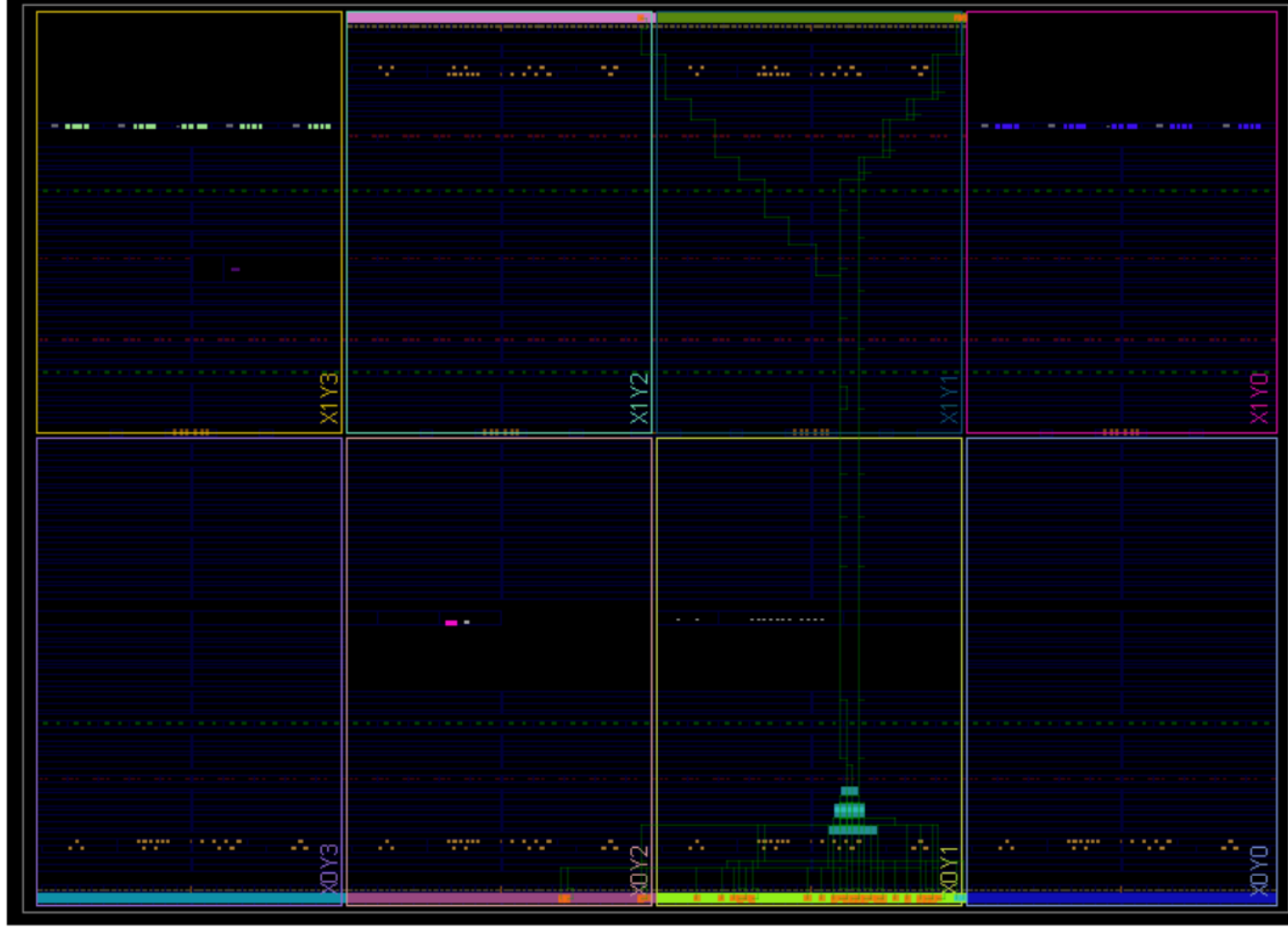


Figure 13 - 8x8 Brauns multiplier FPGA

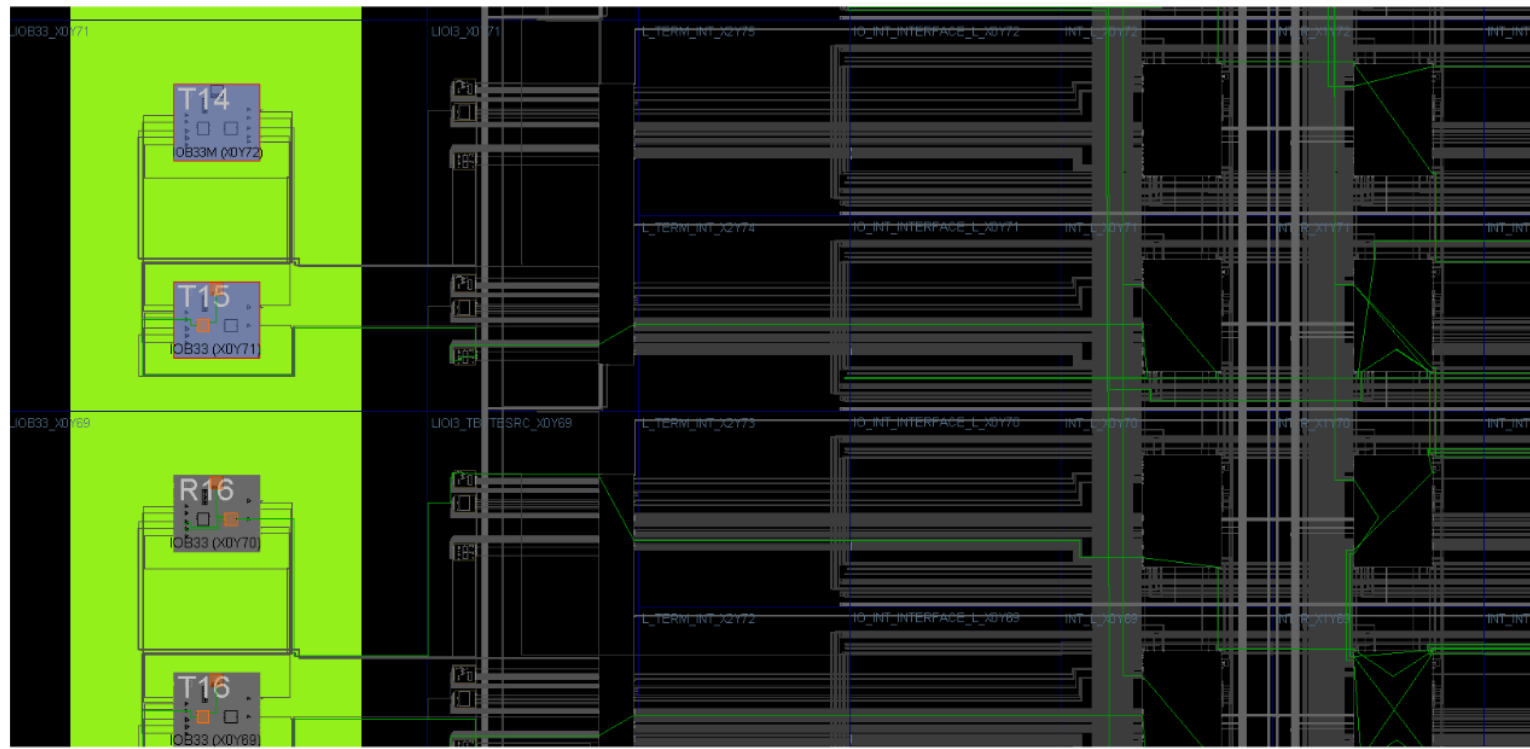


Figure 14 - Some of I/O pins of the FPGA diagram displayed with connections between gates

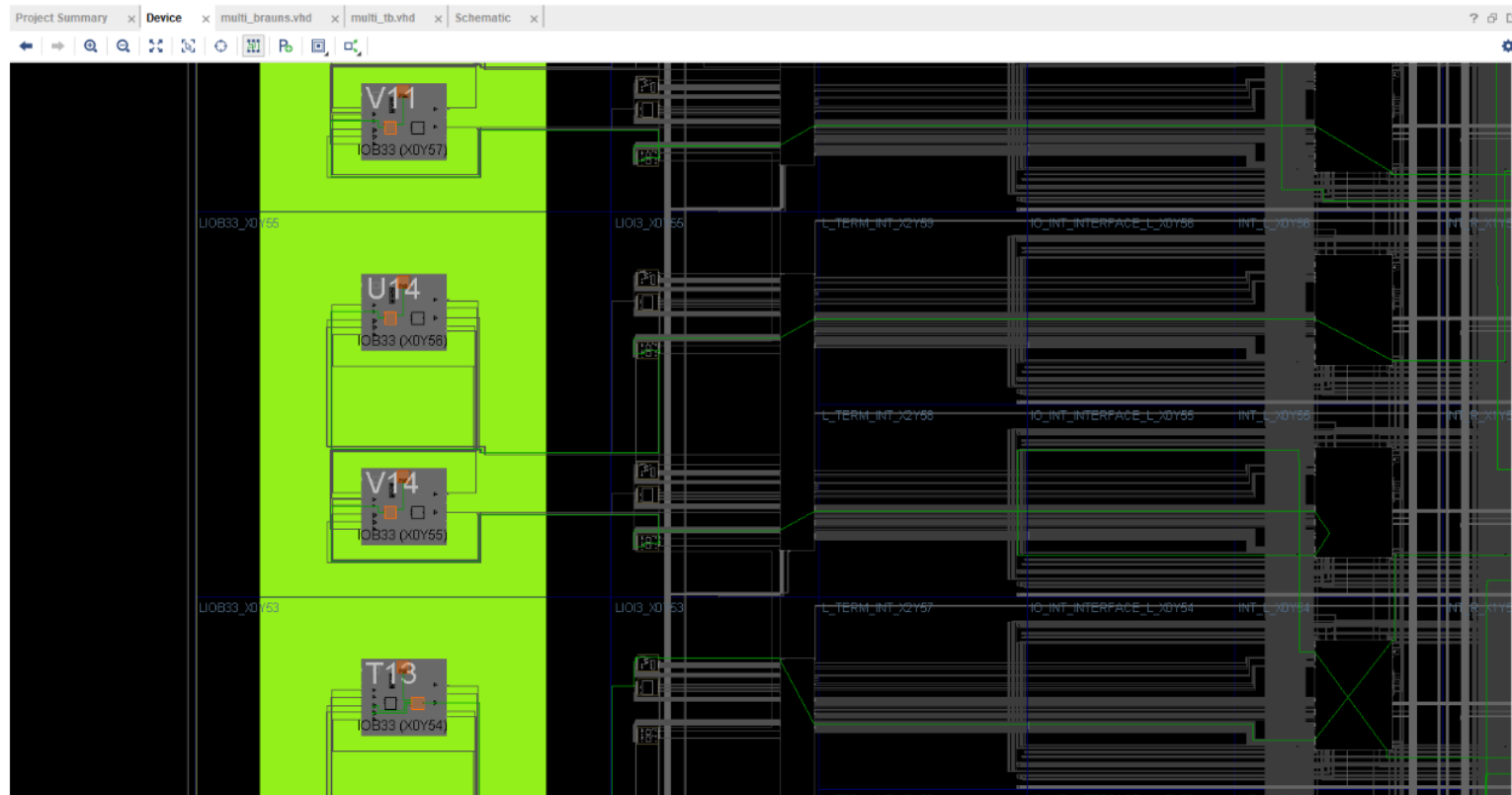


Figure 15 - I/O Pins of the FPGA with interconnections to programmable logic blocks and gates

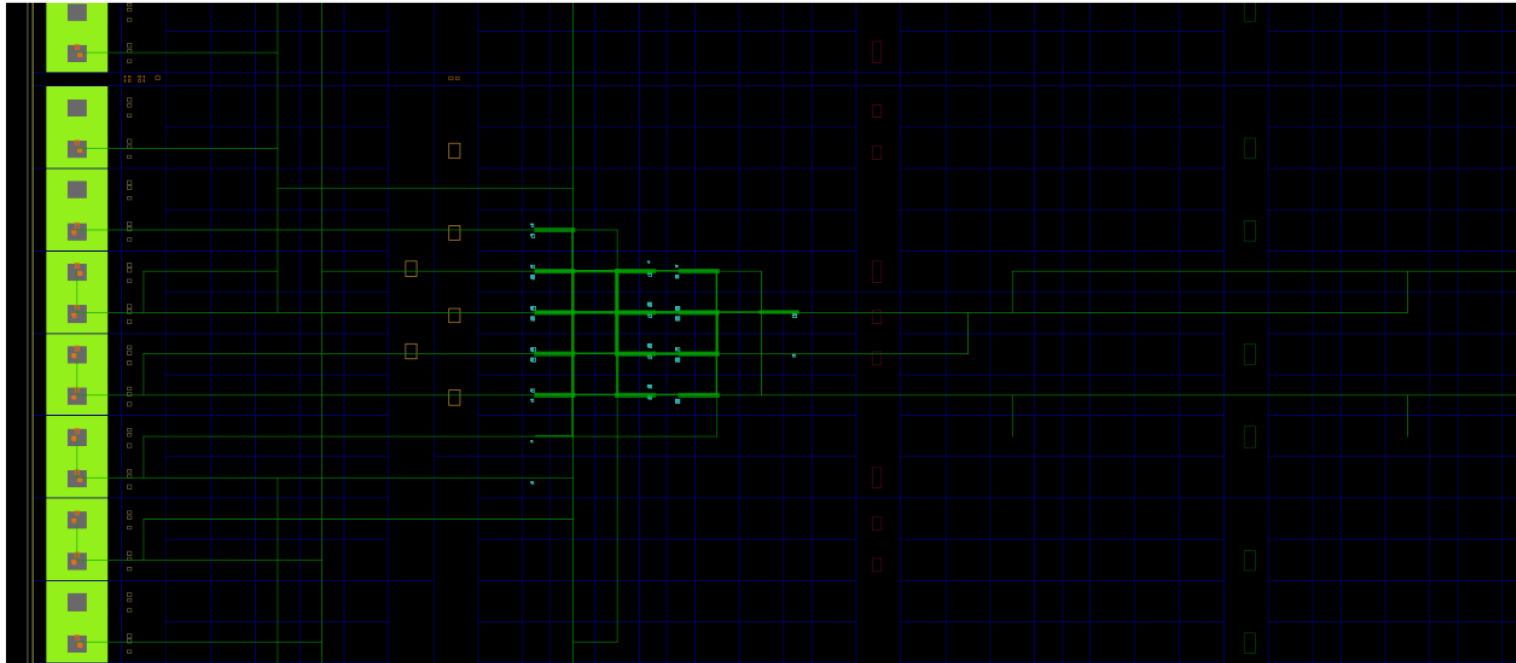


Figure 16 - FPGA diagram demonstrating I/O pins on the left, with inter connections between programmable logic blocks/gates

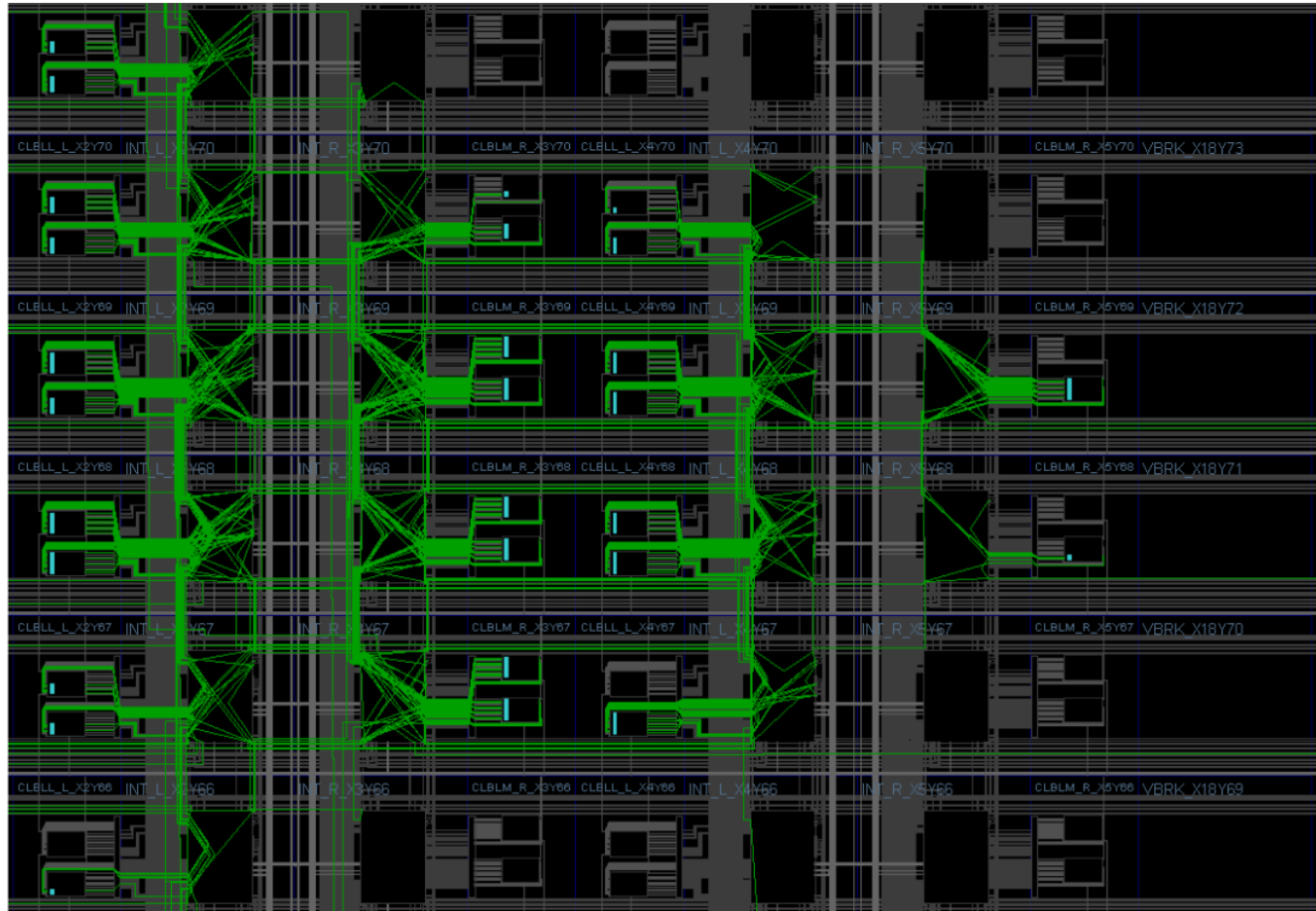


Figure 17 - Area of the FPGA with numerous Programmable logic blocks being used with interconnections between them

6.2 Wallace tree Multiplier FPGA Diagram

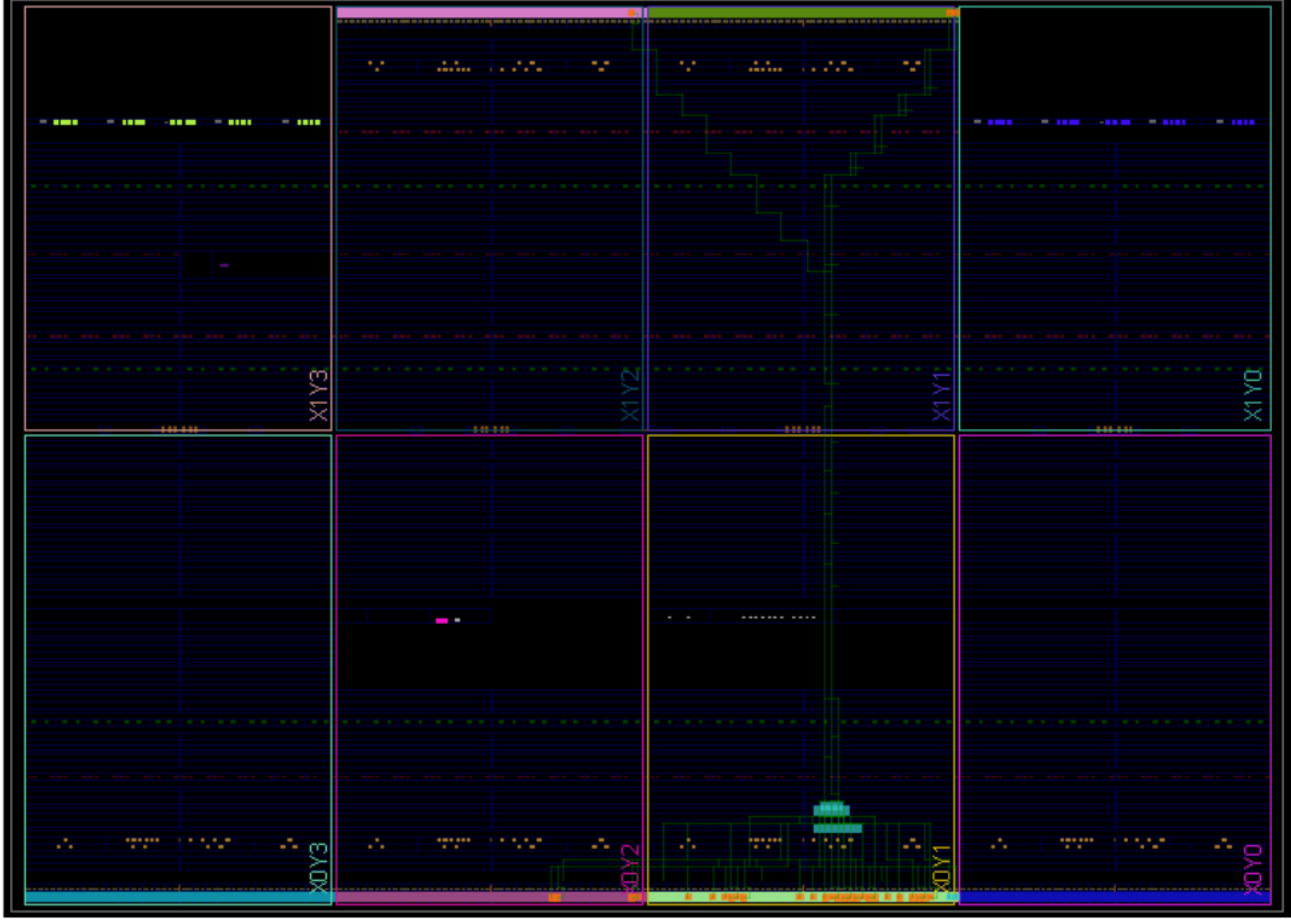


Figure 18 - Wallace tree multiplier FPGA diagram overview

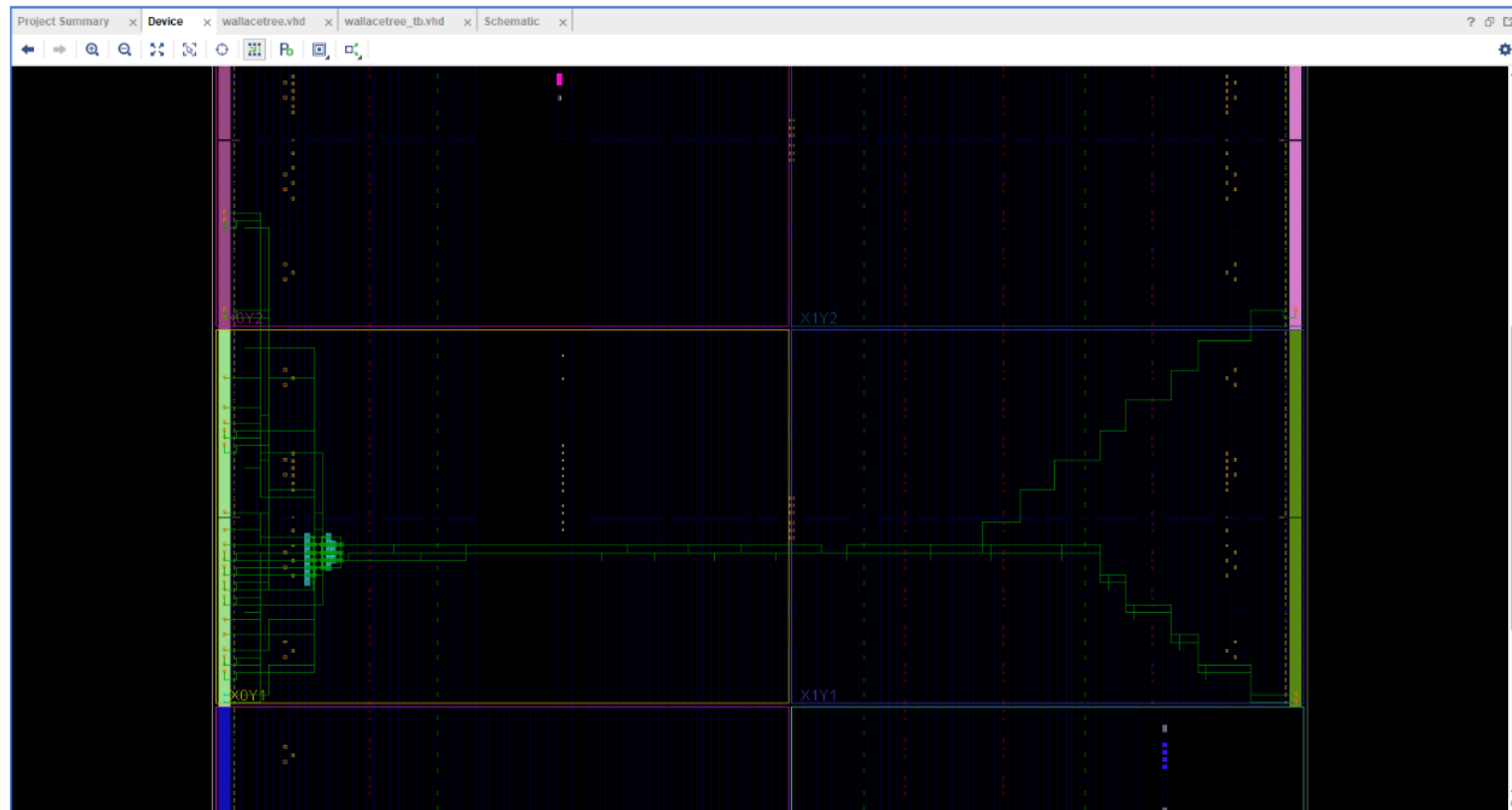


Figure 19 - Wallace tree multiplier FPGA Diagram image zoomed in

Figures 19 to 23 are the FPGA diagram that was given by Vivado when the VHDL circuit that was written for the Wallace tree multiplier was synthesised. Different images of different sections of the FPGA are shown.

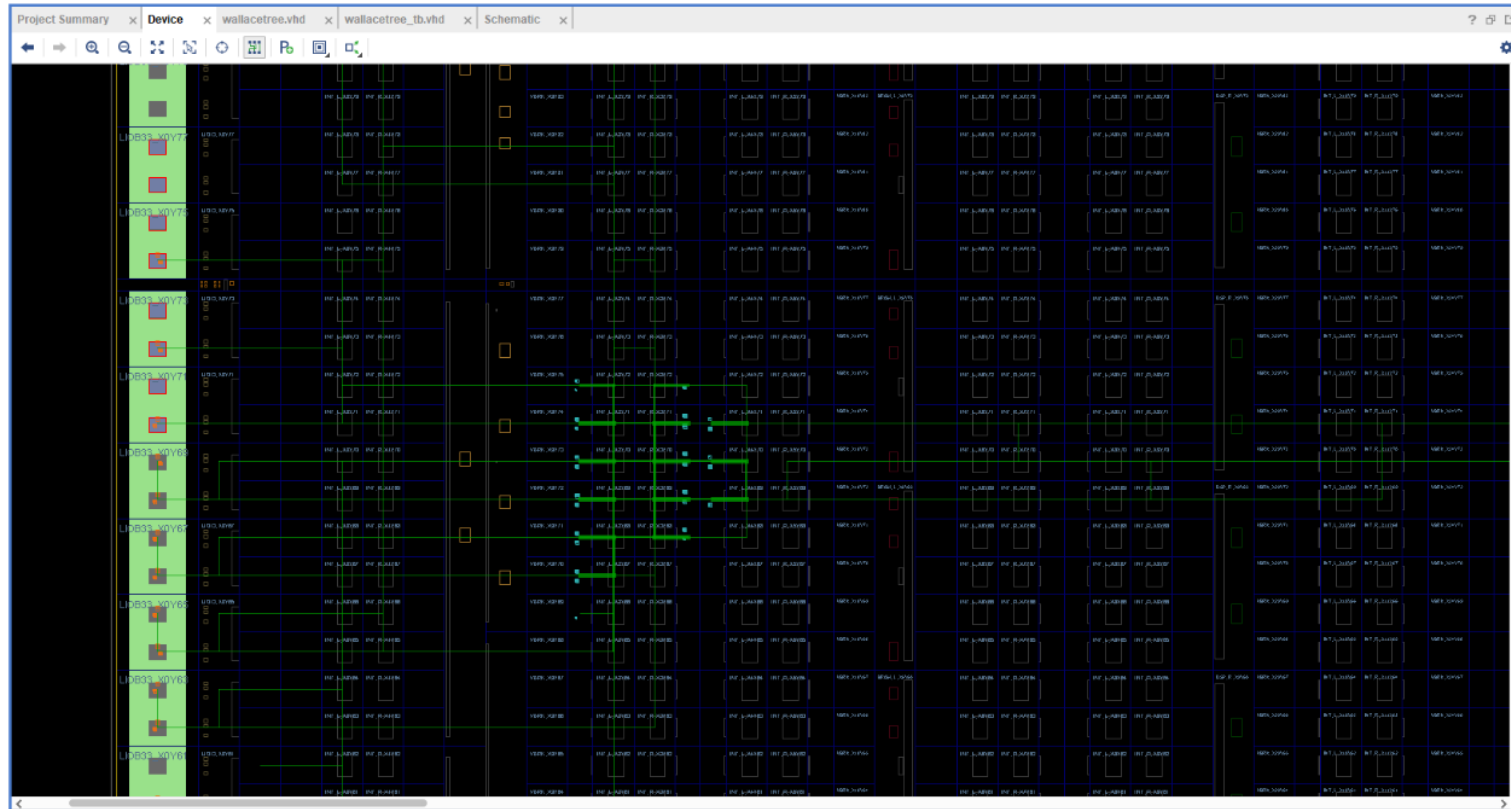


Figure 20 - Wallace tree multiplier FPGA diagram zoomed in on inputs and LUT chips



Figure 21 - Wallace tree multiplier FPGA diagram further zoomed in to Inputs with interconnecting lines between

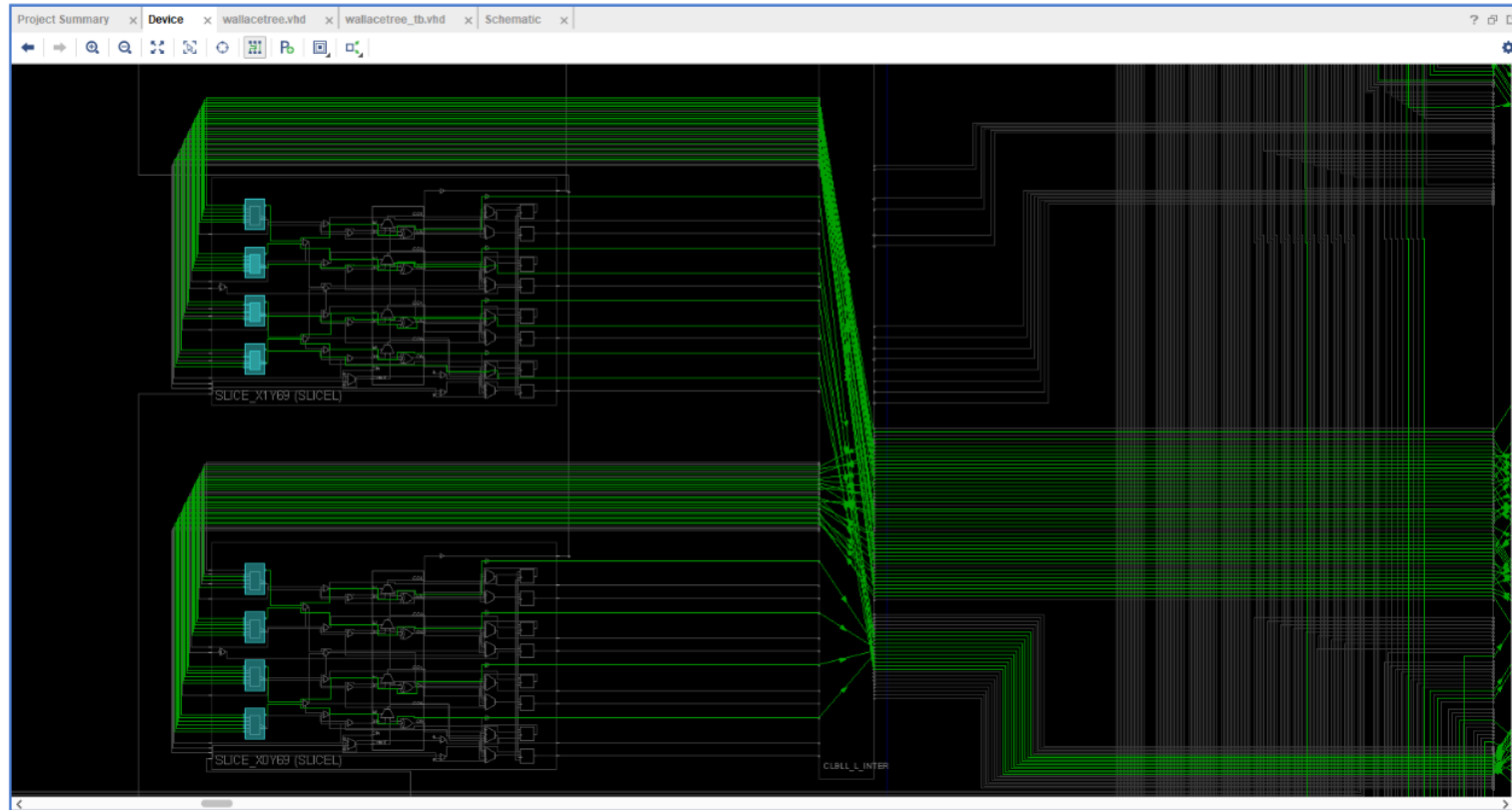


Figure 22 - Wallace tree multiplier FPGA Diagram zoomed in image of LUTs and circuitry with interconnecting switches

7. Timing and schematic diagram

7.1 Braun's Multiplier

7.1.1 Schematic Diagram

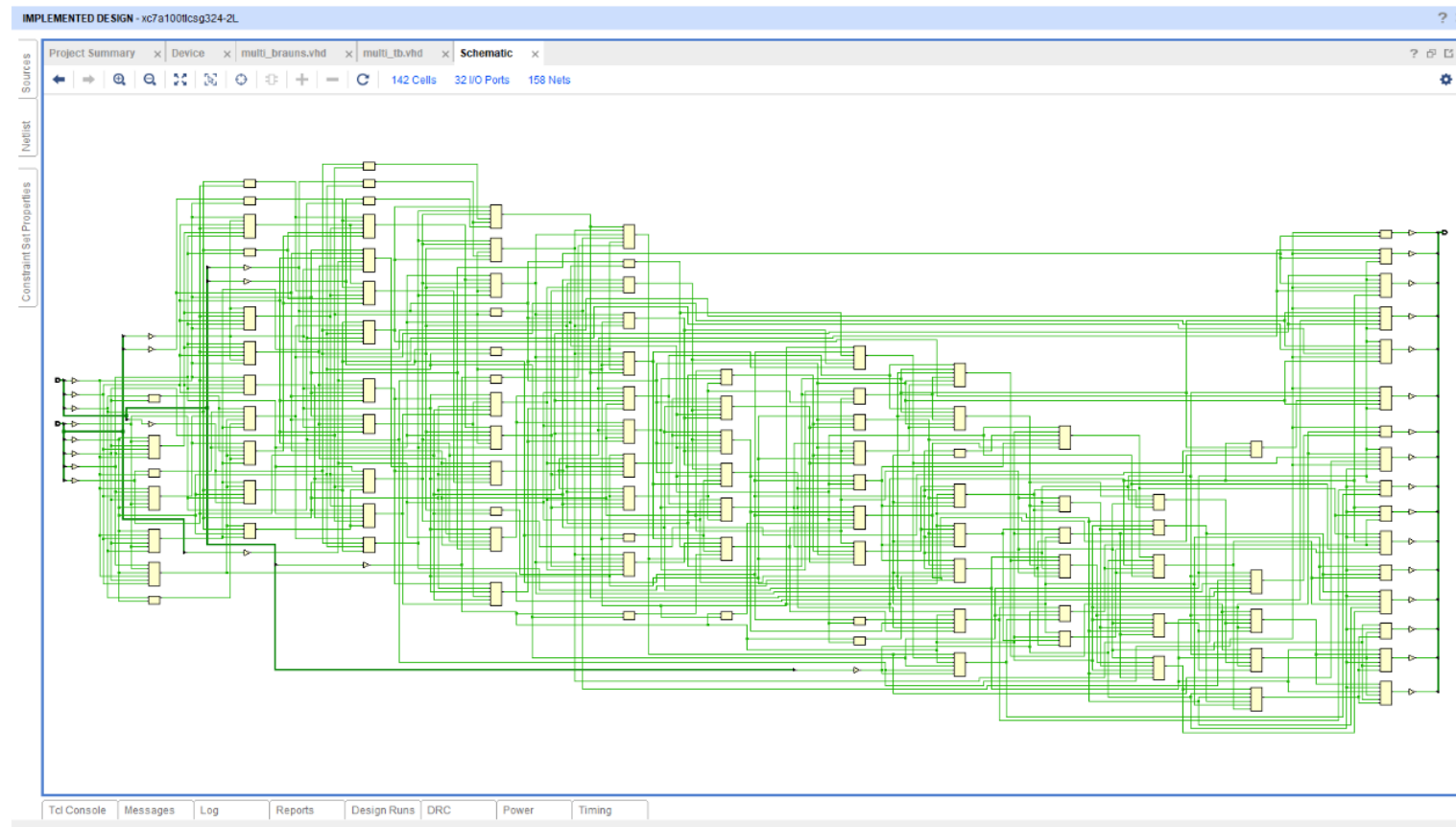


Figure 23 - 8x8 Braun's multiplier structural VHDL code Schematic diagram

Figure's 24 to 26 shows the schematic diagram from the VHDL code written. The schematic's feature NOT gates and Lookup table chips (LUT). Figure 25 shows a close-up of the inputs of the schematic and the surrounding LUTs. Figure 26 shows the output NOT gates of the schematic and the surrounding LUT's that all link together to form the schematic.

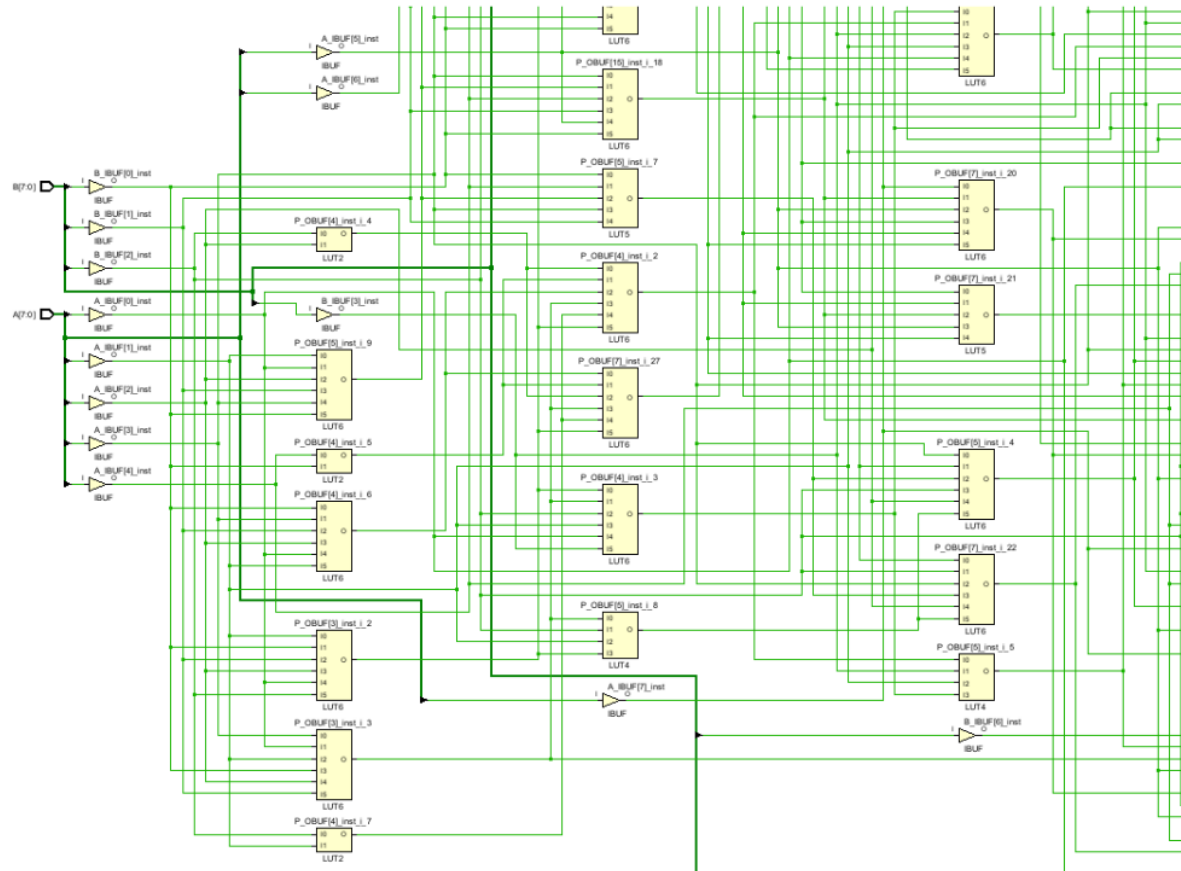


Figure 24 - 8x8 Braun's multiplier VHDL schematic diagram inputs and surrounding LUT chips

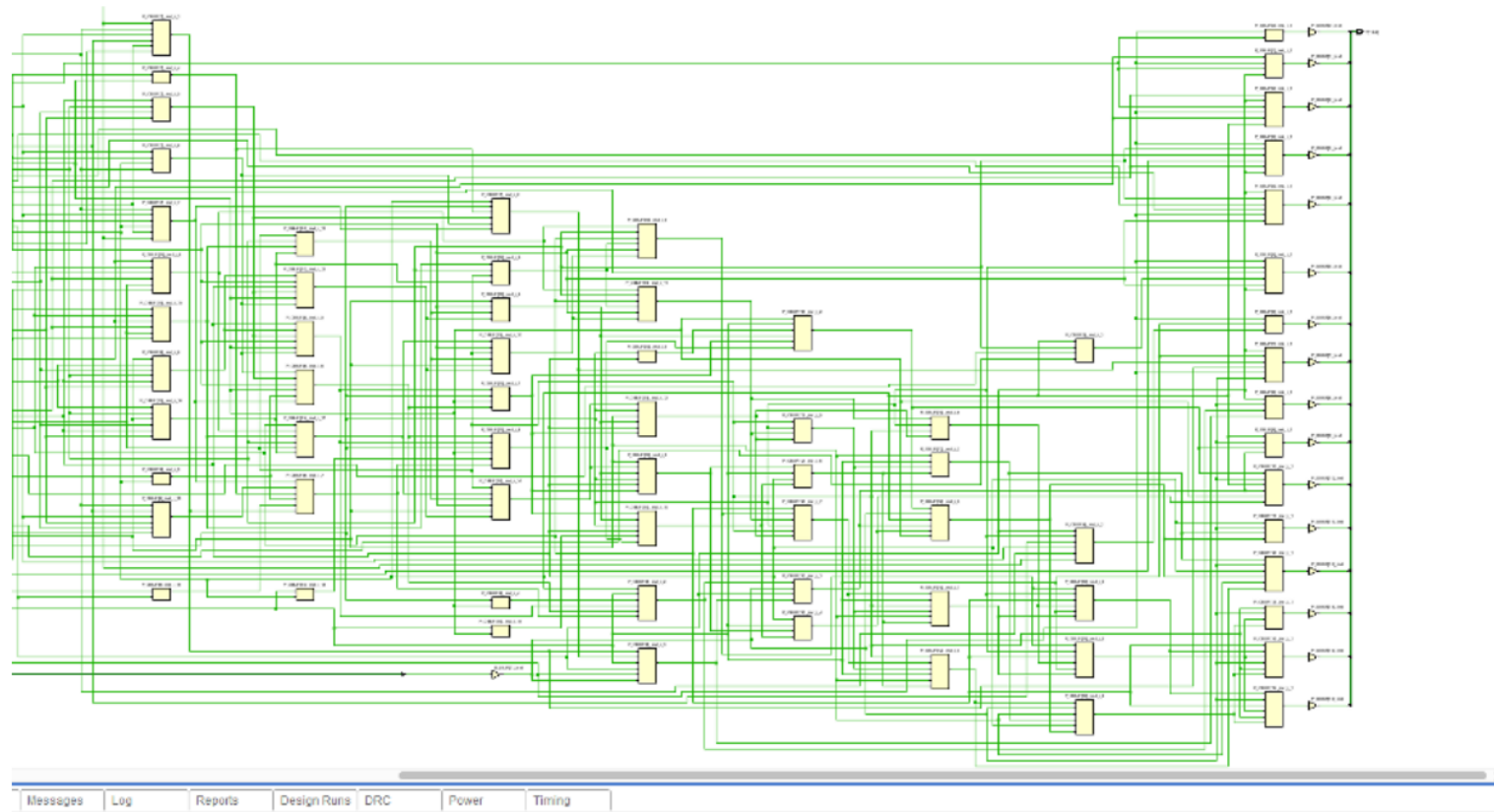


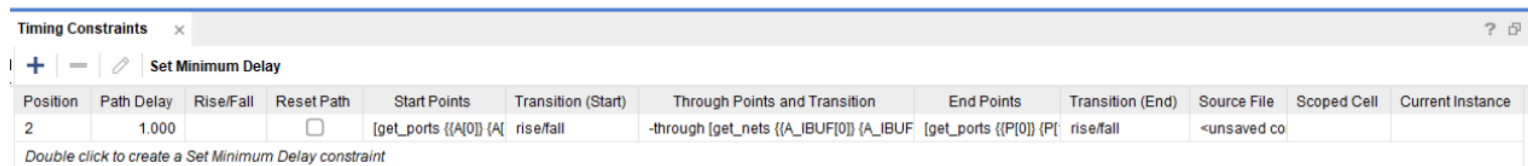
Figure 25 - 8x8 Braun's Multiplier Schematic diagram output NOT gates and LUT's

7.1.2 Timing of Braun's Multiplier

With the use of the logic diagram of Braun's multiplier logic diagram in figure 9 the minimum and maximum delay for the circuit could be calculated and implemented onto the VHDL circuitry within Vivado software. When the path crossed through an AND gate there was a delay applied of 1nS, and when the path crossed through an adder, a delay of 2nS was applied.

The minimum delay of a Braun's multiplier was 1nS, where the path went through 1 AND gate from input to output. This being either from B0 or A0 to P0.

The maximum delay for a Braun's multiplier was 41nS. This was due to the path going through 1 AND gate and 20 adders. Starting at B0 or A1 to P14 or P15. Both of these delays were applied to the VHDL code as shown in figures 27 and 28 below.

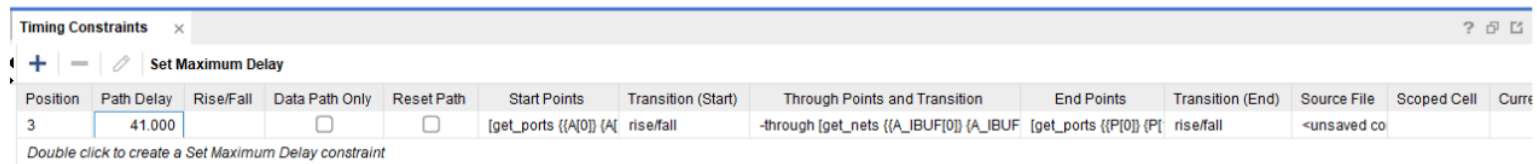


The screenshot shows the 'Timing Constraints' window in Vivado. The 'Set Minimum Delay' tab is active. A table lists a constraint at position 2 with a path delay of 1.000. The start points are [get_ports {{A[0]} {A[1]} rise/fall} and the end points are [get_ports {{P[0]} {P[1]} rise/fall}. The through points are [get_nets {{A_IBUF[0]} {A_IBUF[1]}].

Position	Path Delay	Rise/Fall	Reset Path	Start Points	Transition (Start)	Through Points and Transition	End Points	Transition (End)	Source File	Scoped Cell	Current Instance
2	1.000		☐	[get_ports {{A[0]} {A[1]} rise/fall}		-through [get_nets {{A_IBUF[0]} {A_IBUF[1]}]	[get_ports {{P[0]} {P[1]} rise/fall}		<unsaved co		

Double click to create a Set Minimum Delay constraint

Figure 26 - Setting the minimum delay to the Braun's Multiplier within Vivado



The screenshot shows the 'Timing Constraints' window in Vivado. The 'Set Maximum Delay' tab is active. A table lists a constraint at position 3 with a path delay of 41.000. The start points are [get_ports {{A[0]} {A[1]} rise/fall} and the end points are [get_ports {{P[0]} {P[1]} rise/fall}. The through points are [get_nets {{A_IBUF[0]} {A_IBUF[1]}].

Position	Path Delay	Rise/Fall	Data Path Only	Reset Path	Start Points	Transition (Start)	Through Points and Transition	End Points	Transition (End)	Source File	Scoped Cell	Current Instance
3	41.000		☐	☐	[get_ports {{A[0]} {A[1]} rise/fall}		-through [get_nets {{A_IBUF[0]} {A_IBUF[1]}]	[get_ports {{P[0]} {P[1]} rise/fall}		<unsaved co		

Double click to create a Set Maximum Delay constraint

Figure 27 - Setting Maximum Delay to the Braun's Multiplier within Vivado

The delays got added to the VHDL build of a Braun's multiplier within Vivado by choosing the ports that the path starts with, the through points and the endpoints as shown in figure's 29 and 30.

Edit Set Minimum Delay X

Specify minimum delay for timing paths.

Specify path delay: ns

Targets Options

Start Points

From: {B[1]} {B[2]} {B[3]} {B[4]} {B[5]} {B[6]} {B[7]]} ... Transition rise/fall

Through Points

Through: {I00 S_101 S_102 S_103 S_104]]} ... Transition rise/fall +

End Points

To: {P[10]] {P[11]] {P[12]] {P[13]] {P[14]] {P[15]]}} ... Transition rise/fall

Command: 3]] {P[4]] {P[5]] {P[6]] {P[7]] {P[8]] {P[9]] {P[10]] {P[11]] {P[12]] {P[13]] {P[14]] {P[15]]}} 1.0

? Reference Reset to Defaults OK Cancel

Figure 28 - Minimum Delay being Set by choosing the points the Path from Input to output take

Edit Set Maximum Delay X

Specify maximum delay for timing paths.

Specify path delay: ns

Targets Options

Start Points

From: {B[1]} {B[2]} {B[3]} {B[4]} {B[5]} {B[6]} {B[7]]} ... Transition rise/fall

Through Points

Through: {I00 S_101 S_102 S_103 S_104]]} ... Transition rise/fall +

End Points

To: {P[10]] {P[11]] {P[12]] {P[13]] {P[14]] {P[15]]}} ... Transition rise/fall

Command: 3]] {P[4]] {P[5]] {P[6]] {P[7]] {P[8]] {P[9]] {P[10]] {P[11]] {P[12]] {P[13]] {P[14]] {P[15]]}} 41.0

? Reference Reset to Defaults OK Cancel

Figure 29 - Maximum Delay being set by choosing the points the path from input to output take

7.2 Wallace Tree Multiplier

7.2.1 Schematic Diagram

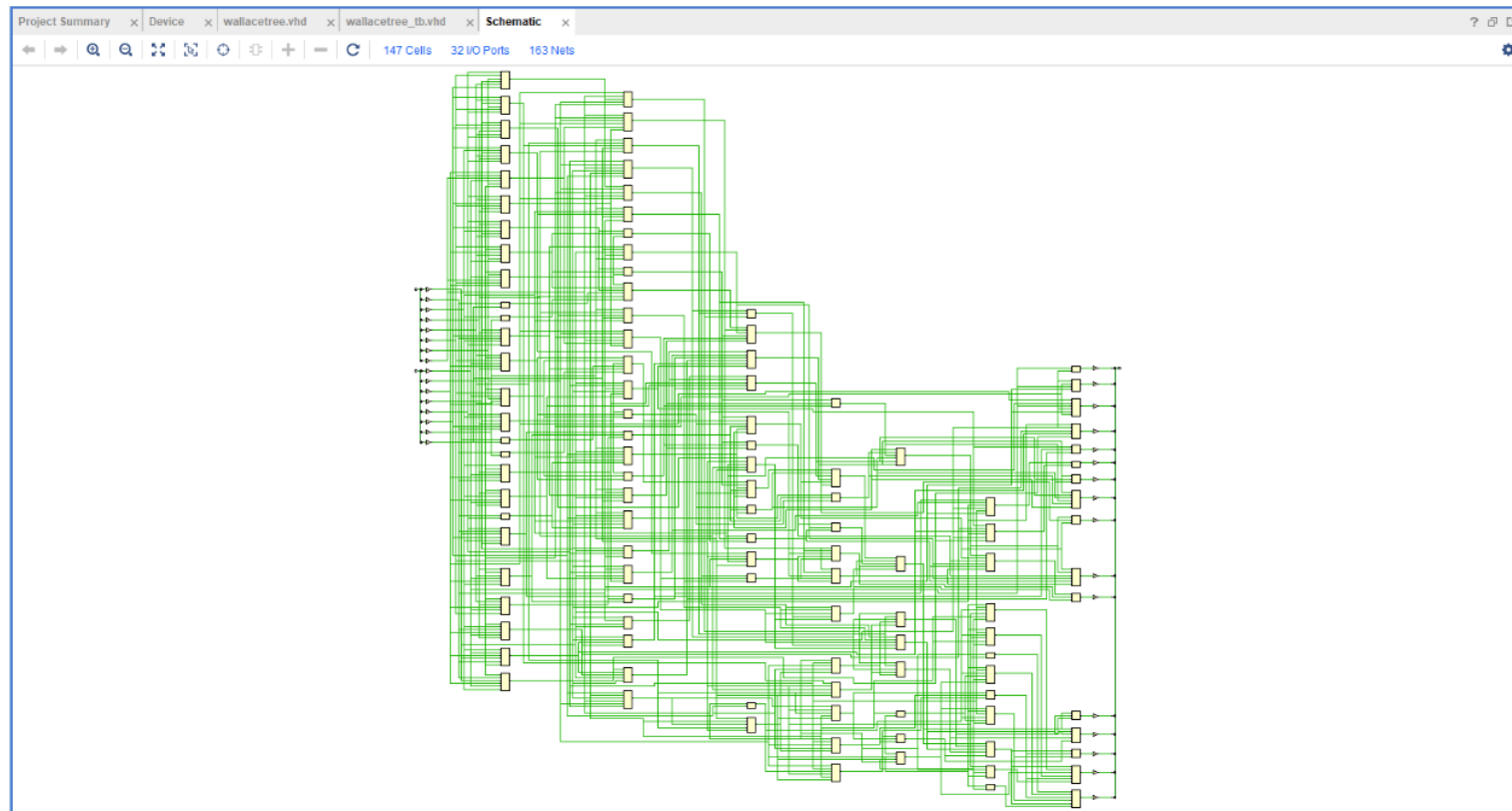


Figure 30 - VHDL Schematic of the 8x8 Wallace tree multiplier

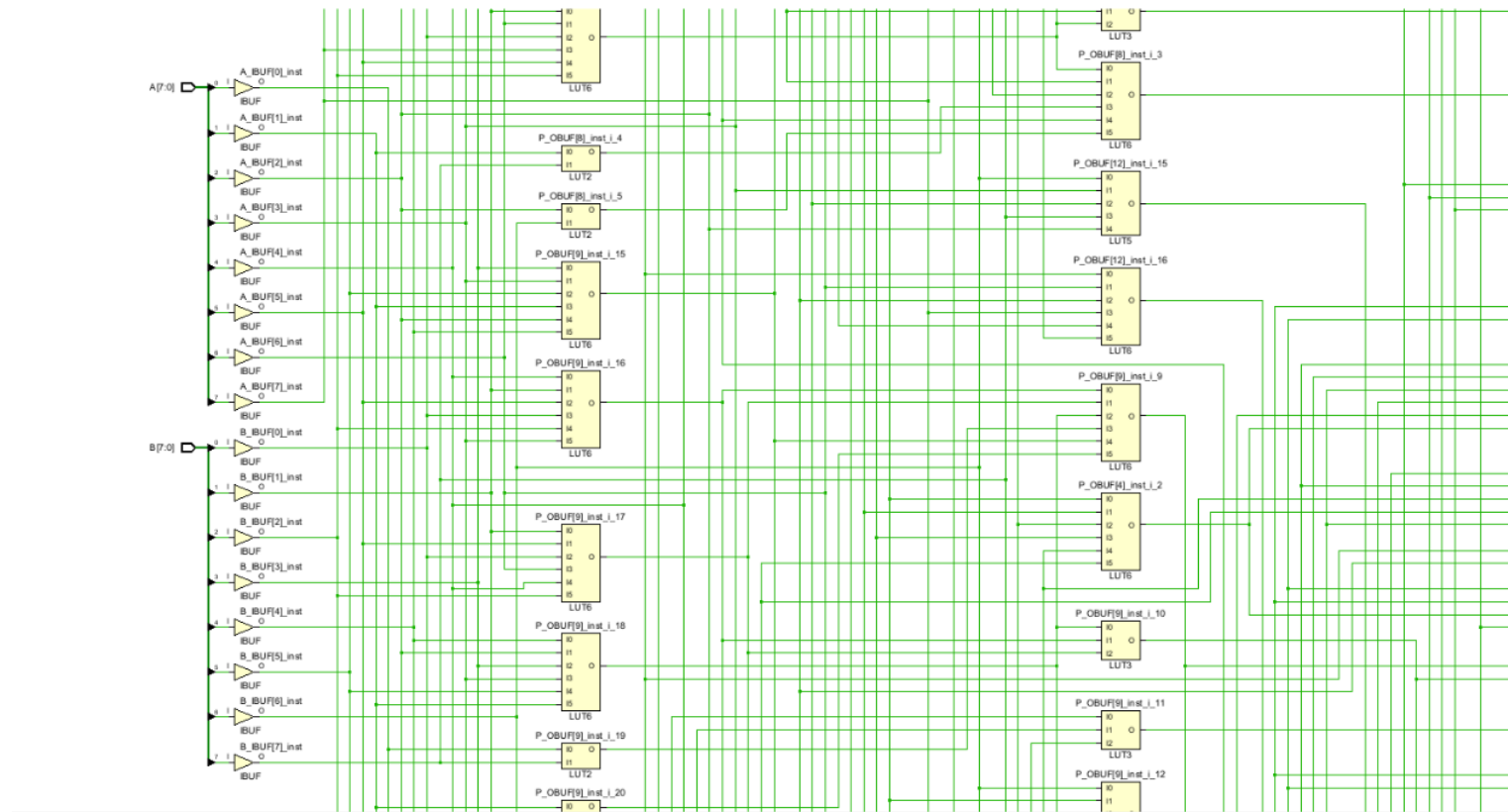


Figure 31 - Schematic from the VHDL code for an 8x8 Wallace tree multiplier showing inputs and LUTs

The VHDL code written for an 8x8 Wallace tree multiplier allowed the simulation to be run and synthesised, the program then produced a schematic as visible in figures 31 to 33. Figure 32 is a close-up of the inputs to the schematic which are A7-A0 and B7-B0. Figure 33 shows some of the outputs created by Vivado for the schematic of the Wallace tree multiplier.

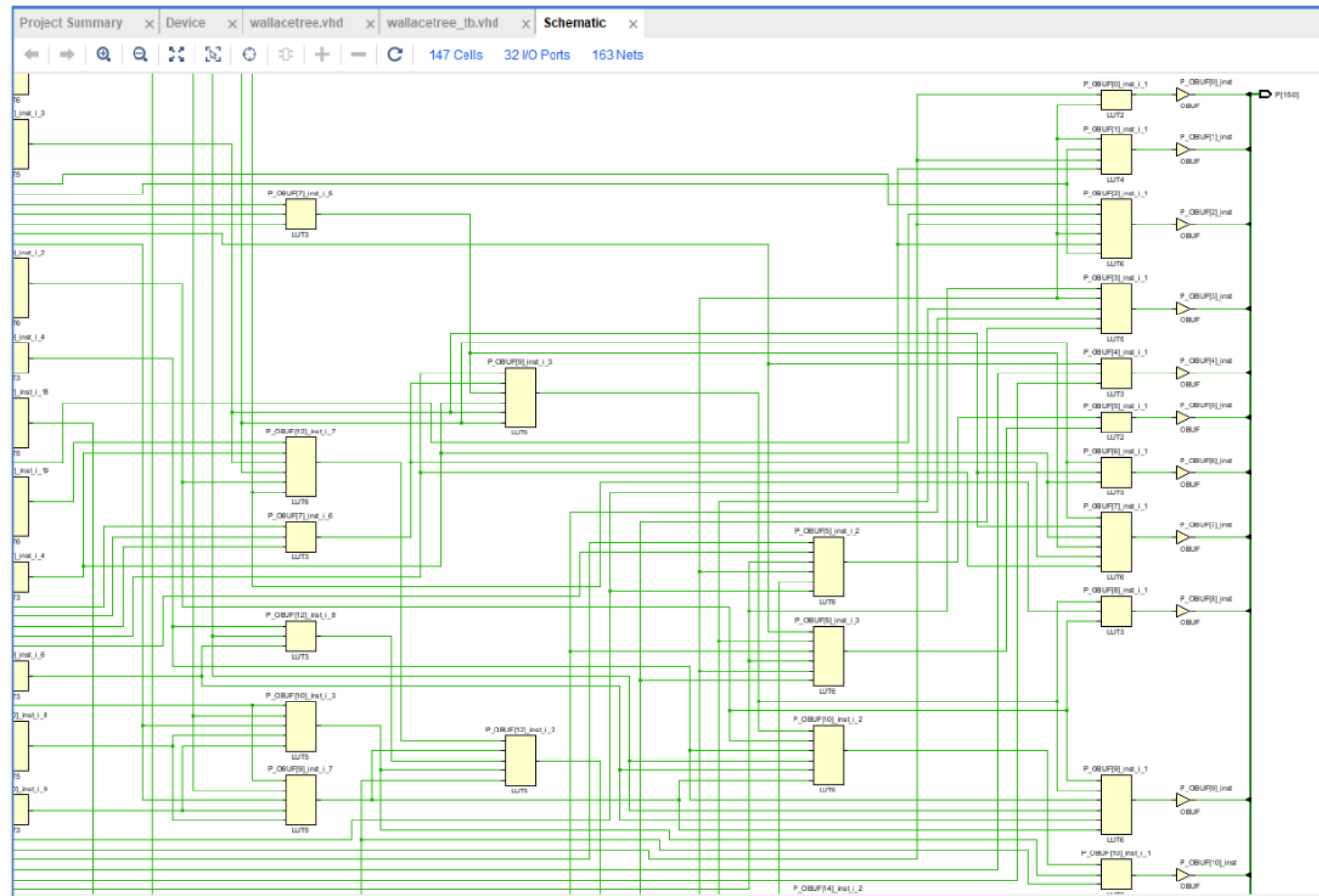


Figure 32 - Schematic diagram produced by Vivado using a VHDL code for 8x8 Wallace tree multiplier showing part of the outputs

7.2.2 Timings of Wallace Tree Multiplier

Timing delays were also applied to the Wallace tree multiplier, using the same values as the Braun's Multiplier where when the path crossed through an AND gate there was a delay applied of 1nS, and when the path crossed through an adder, a delay of 2nS was applied.

Using the Logic Diagram circuit for a Wallace tree multiplier in Figure 11 the shortest delay was 1nS, with the path passing through 1 AND gate from B0 or A0 to P0.

The maximum delay was 31nS where the path went through 1 AND gate and 15 adders. Starting at either B1, B2 or B0 and finishing at P15.

The maximum and minimum delays were then applied to the Circuit on Vivado.

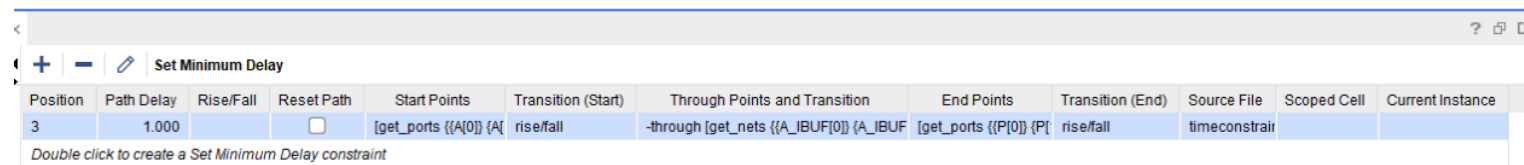


Figure 33 - Minimum delay being set on the Wallace Tree Multiplier

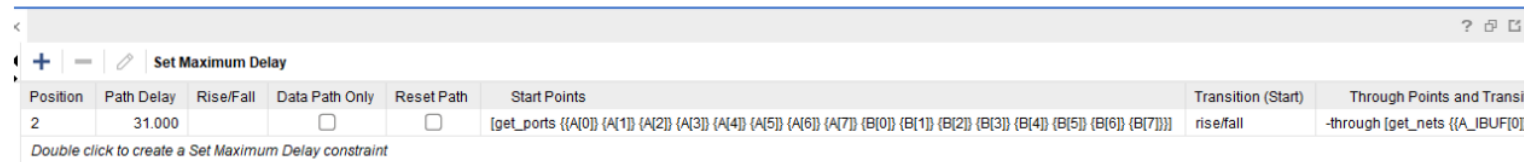


Figure 34 - Maximum delay being set on the Wallace tree multiplier

The Minimum and maximum delays we're also set by choosing which points the paths start at, which are the through points and what are the endpoints. This is shown in figure 36 and 37.

Edit Set Minimum Delay

Specify minimum delay for timing paths.

Specify path delay: ns

Targets Options

Start Points

From: {B[1]} {B[2]} {B[3]} {B[4]} {B[5]} {B[6]} {B[7]} ... Transition rise/fall

Through Points

Through: {} {P_OBUF[15_inst_i_14_n_0]} ... Transition rise/fall +

End Points

To: {} {P[10]} {P[11]} {P[12]} {P[13]} {P[14]} {P[15]} ... Transition rise/fall

Command: {3} {P[4]} {P[5]} {P[6]} {P[7]} {P[8]} {P[9]} {P[10]} {P[11]} {P[12]} {P[13]} {P[14]} {P[15]} 1.0

? Reference Reset to Defaults OK Cancel

Figure 36 - Minimum delay being set for the Wallace Tree Multiplier by choosing the Input, through points and outputs for the path

Edit Set Maximum Delay

Specify maximum delay for timing paths.

Specify path delay: ns

Targets Options

Start Points

From: {B[1]} {B[2]} {B[3]} {B[4]} {B[5]} {B[6]} {B[7]} ... Transition rise/fall

Through Points

Through: {} {P_OBUF[15_inst_i_14_n_0]} ... Transition rise/fall +

End Points

To: {} {P[10]} {P[11]} {P[12]} {P[13]} {P[14]} {P[15]} ... Transition rise/fall

Command: {} {P[4]} {P[5]} {P[6]} {P[7]} {P[8]} {P[9]} {P[10]} {P[11]} {P[12]} {P[13]} {P[14]} {P[15]} 31.0

? Reference Reset to Defaults OK Cancel

Figure 35 - Maximum delay being set for the Wallace tree Multiplier by choosing the input, through points and outputs for the path

8. Simulation Results

8.1 Braun's Multiplier

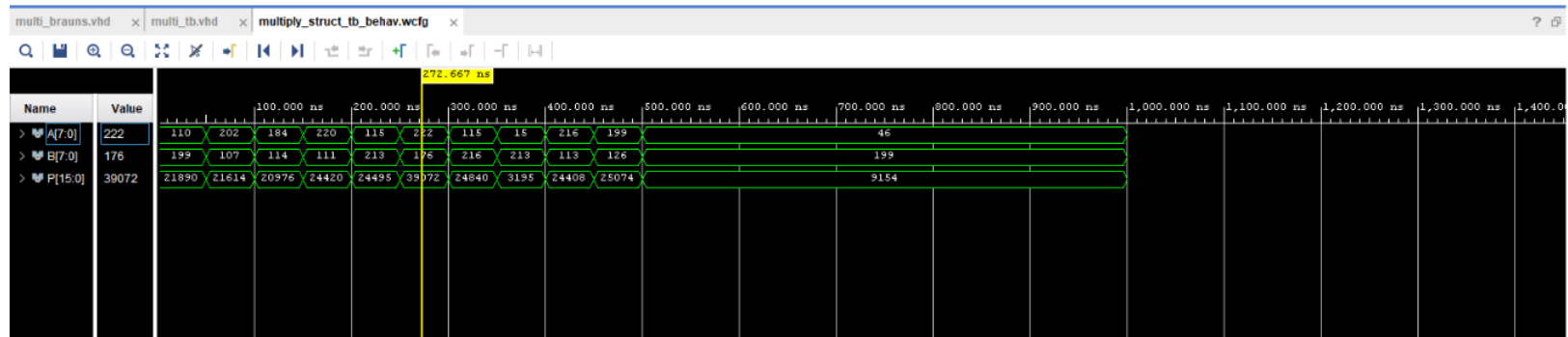


Figure 37 - 8x8 Braun's multiplier multiplication results from test bench

16

Figure 38 is the results from the equations written into the test bench to test whether Braun's multiplier was working as it should from the equations, the correct answers are given by the output P when the inputs are multiplied together. The inputs and outputs are shown as unsigned decimals. The test bench's equations and code can be found in In the appendix 2. Both the multipliers used the same sample equations to keep consistency between them both for the comparisons.

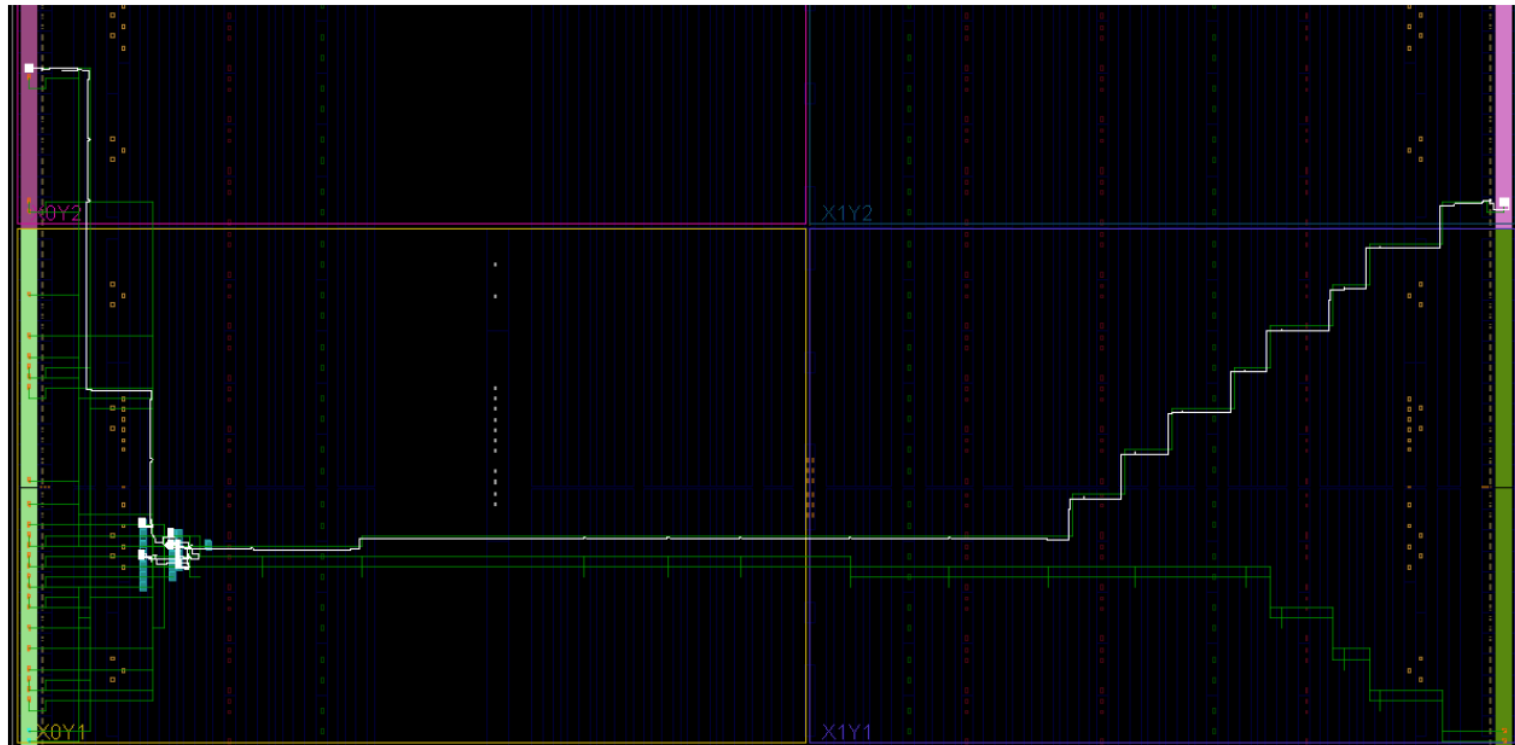


Figure 38 - FPGA Longest Delay Path on the FPGA board

Figure's 39 and 40 Show the minimum and maximum delay paths on the FPGA circuitry after the delays were implemented on Vivado.

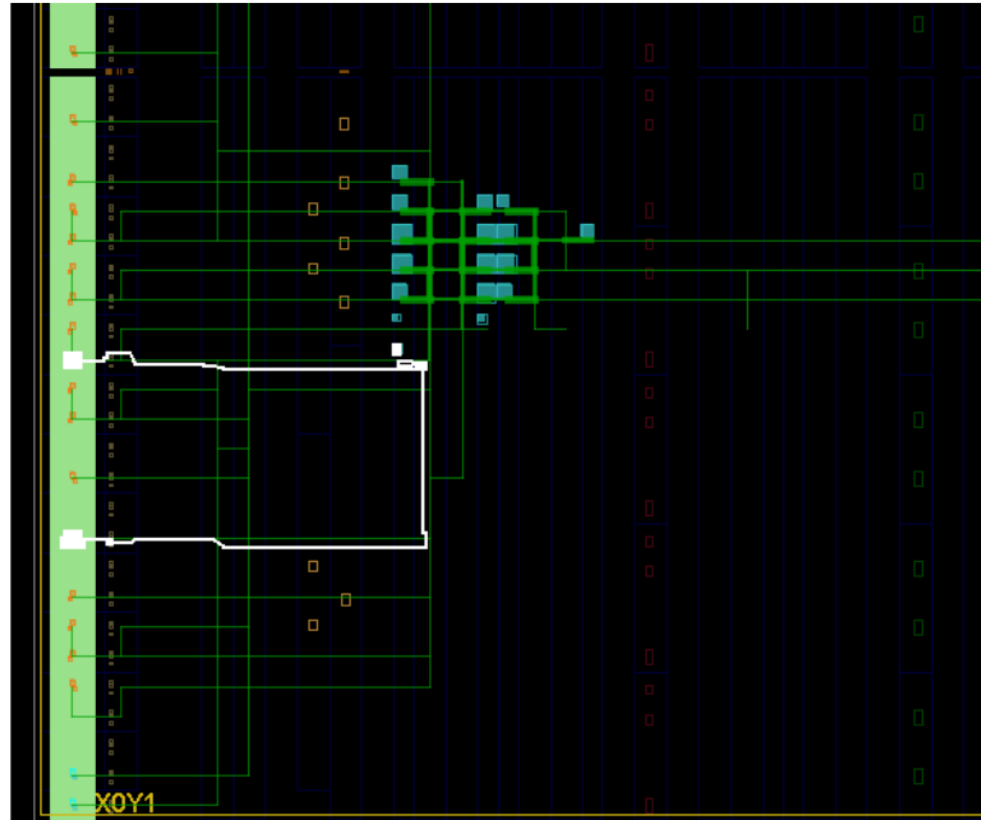


Figure 39 - Brauns Multiplier FPGA minimum Delay path on the FPGA Circuit

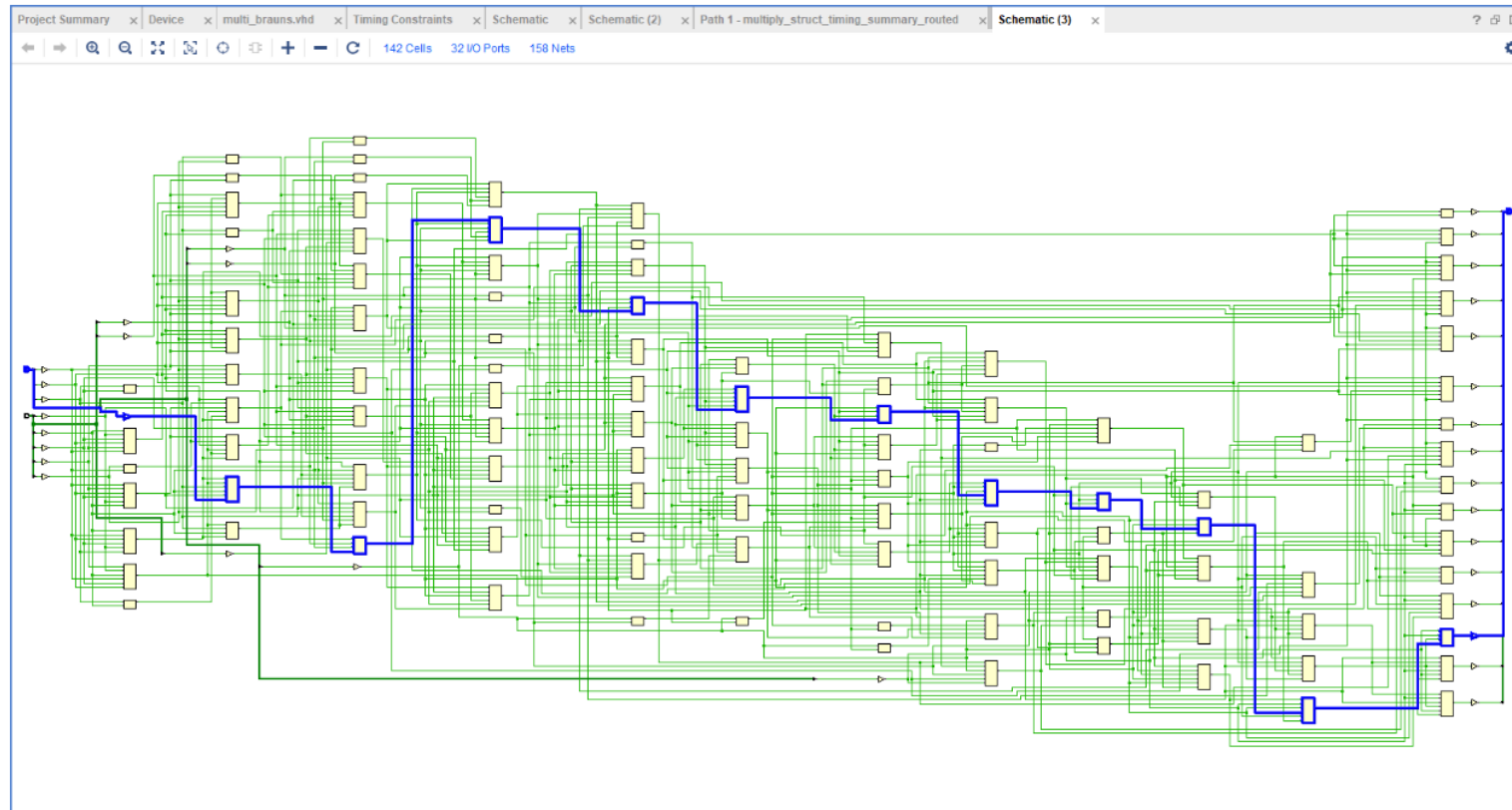


Figure 40 – Braun's Multiplier longest path delay on the Schematic diagram of VHDL Circuit

Figure 41 and 42 Show the schematic diagram of Braun's multiplier within Vivado. The minimum and maximum paths are shown within these images.

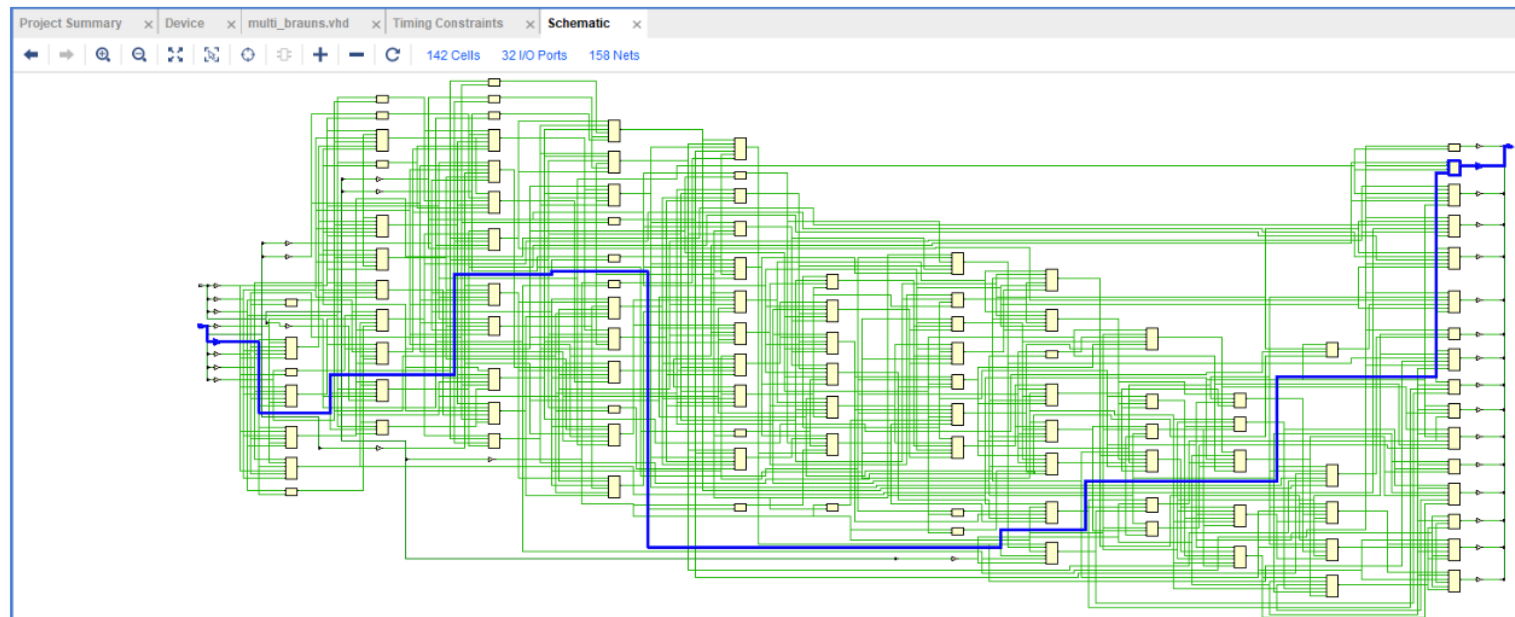


Figure 41 - Braun's Multiplier Schematic diagram with Minimum delay path of the circuit

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	19.433	13	19	B[3]	P[13]	21.567	6.220	15.347	41.0	input port clock		MaxDelay Path 41.000ns	0.000
Path 2	19.617	13	19	B[3]	P[15]	21.383	6.413	14.970	41.0	input port clock		MaxDelay Path 41.000ns	0.000
Path 3	20.262	13	19	B[3]	P[14]	20.738	6.428	14.310	41.0	input port clock		MaxDelay Path 41.000ns	0.000
Path 4	20.604	12	19	B[3]	P[11]	20.396	6.071	14.325	41.0	input port clock		MaxDelay Path 41.000ns	0.000
Path 5	20.835	12	19	B[3]	P[12]	20.165	6.070	14.095	41.0	input port clock		MaxDelay Path 41.000ns	0.000
Path 6	22.102	10	19	B[3]	P[9]	18.898	6.003	12.895	41.0	input port clock		MaxDelay Path 41.000ns	0.000

Figure 42 - Braun's Multiplier Longest delay paths of the VHDL Circuit

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 11	1.860	3	22	A[1]	P[1]	2.860	1.794	1.065	1.0	input port clock		MinDelay Path 1.000ns	0.000
Path 12	1.877	3	22	A[1]	P[3]	2.877	1.776	1.101	1.0	input port clock		MinDelay Path 1.000ns	0.000
Path 13	2.060	3	22	A[1]	P[5]	3.060	1.777	1.283	1.0	input port clock		MinDelay Path 1.000ns	0.000
Path 14	2.097	3	22	A[1]	P[7]	3.097	1.772	1.325	1.0	input port clock		MinDelay Path 1.000ns	0.000
Path 15	2.130	3	22	A[1]	P[8]	3.130	1.846	1.283	1.0	input port clock		MinDelay Path 1.000ns	0.000
Path 16	2.185	3	22	A[1]	P[4]	3.185	1.756	1.429	1.0	input port clock		MinDelay Path 1.000ns	0.000
Path 17	2.197	3	16	B[1]	P[2]	3.197	1.794	1.403	1.0	input port clock		MinDelay Path 1.000ns	0.000

Figure 43 - Braun's Multiplier Shortest path delays of the VHDL circuit

In the figure's 43 and 44 above Braun's multiplier longest paths of the circuit and the shortest delayed paths of the circuit can be seen.

Summary	
Name	Path 1
Slack	19.433ns
Source	B[3] (input port)
Destination	P[13]
Path Group	**default**
Path Type	Max at Slow Process Corner
Requirement	41.000ns (MaxDelay Path 41.000ns)
Data Path Delay	21.567ns (logic 6.220ns (28.840%) route 15.347ns (71.160%))
Logic Levels	13 (IBUF=1 LUT4=6 LUT6=5 OBUF=1)
Output Delay	0.000ns
Timing Exception	MaxDelay Path 41.000ns

Figure 45 - Longest Delay Path of the Braun's Multiplier in VHDL on Vivado

Data Path				
Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
	(r) 0.000	0.000	Site: H6	B[3]
net (fo=0)	0.000	0.000		B[3]
IBUF (Prop_ibuf I O)	(r) 1.003	1.003	Site: H6	B_IBUF[3]_inst/O
net (fo=19, routed)	5.180	6.182		B_IBUF[3]
LUT6 (Pro...t6 I5 O)	(r) 0.148	6.330	Site: S...X0Y68	P_OBUF[4]_inst_i_3/O
net (fo=3, routed)	0.701	7.031		C_54
LUT4 (Pro...t4 I3 O)	(r) 0.170	7.201	Site: S...X1Y68	P_OBUF[5]_inst_i_5/O
net (fo=5, routed)	0.875	8.076		C_16
LUT6 (Pro...t6 I0 O)	(r) 0.362	8.438	Site: S...X4Y67	P_OBUF[7]_inst_i_9/O
net (fo=5, routed)	0.801	9.239		C_24
LUT4 (Pro...t4 I3 O)	(r) 0.160	9.399	Site: S...X4Y67	P_OBUF[7]_inst_i_6/O
net (fo=4, routed)	0.492	9.892		C_25
LUT6 (Pro...t6 I5 O)	(r) 0.378	10.270	Site: S...X4Y69	P_OBUF[10]_inst_i_12/O
net (fo=3, routed)	0.849	11.118		C_27
LUT4 (Pro...t4 I3 O)	(r) 0.170	11.288	Site: S...X4Y68	P_OBUF[10]_inst_i_6/O
net (fo=4, routed)	0.791	12.079		S_88
LUT6 (Pro...t6 I3 O)	(r) 0.362	12.441	Site: S...X3Y69	P_OBUF[15]_inst_i_12/O
net (fo=3, routed)	0.706	13.147		C_36

Figure 47 - Longest delayed path's route from input to output of the Braun's Multiplier

LUT4 (Pro...t4 I3 O)	(r) 0.148	13.295	Site: S...X2Y69	P_OBUF[12]_inst_i_5/O
net (fo=3, routed)	0.722	14.017		S_96
LUT4 (Pro...t4 I2 O)	(r) 0.148	14.165	Site: S...X2Y70	P_OBUF[12]_inst_i_2/O
net (fo=3, routed)	0.559	14.725		S_102
LUT6 (Pro...t6 I3 O)	(r) 0.148	14.873	Site: S...X0Y71	P_OBUF[15]_inst_i_4/O
net (fo=3, routed)	0.497	15.369		C_52
LUT4 (Pro...t4 I3 O)	(r) 0.148	15.517	Site: S...X0Y71	P_OBUF[13]_inst_i_1/O
net (fo=1, routed)	3.175	18.692		P_OBUF[13]
OBUF (Pr...buf I O)	(r) 2.875	21.567	Site: J13	P_OBUF[13]_inst/O
net (fo=0)	0.000	21.567		P[13]
			Site: J13	P[13]
Arrival Time		21.567		

Figure 46 - Longest delayed paths route from input to output of the Braun's multiplier continued

Figure's 46 to 48 show the longest delayed paths I/O's and also the LUT and chips in goes through on the way from start to finish.

Figure's 49 and 50 show the shortest delayed paths I/O's and the LUT and chips it goes through on the way from start to finish.

Summary	
Name	↳ Path 11
Slack (Hold)	1.860ns
Source	⏏ A[1] (input port)
Destination	⏏ P[1]
Path Group	**default**
Path Type	Min at Fast Process Corner
Requirement	1.000ns (MinDelay Path 1.000ns)
Data Path Delay	2.860ns (logic 1.794ns (62.744%) route 1.065ns (37.256%))
Logic Levels	3 (IBUF=1 LUT4=1 OBUF=1)
Output Delay	0.000ns
Timing ...ception	MinDelay Path 1.000ns

Figure 49 - Shortest Delay of the Braun's multiplier on VHDL code

Data Path				
Delay Type	Incr (ns)	Path ...	Location	Netlist Resource(s)
	(r) 0.000	0.000	Site: U18	⏏ A[1]
net (fo=0)	0.000	0.000		↗ A[1]
IBUF (Prop. ibuf I O)	(r) 0.441	0.441	Site: U18	⏏ A_IBUF[1]_inst/O
net (fo=22, routed)	0.452	0.893		↗ A_IBUF[1]
LUT4 (Pro...t4 I3 O)	(r) 0.062	0.955	Site: S..._X0Y65	⏏ P_OBUF[1]_inst_i_1/O
net (fo=1, routed)	0.613	1.568		↗ P_OBUF[1]
OBUF (Pr...buf I O)	(r) 1.292	2.860	Site: V12	⏏ P_OBUF[1]_inst/O
net (fo=0)	0.000	2.860		↗ P[1]
			Site: V12	⏏ P[1]
Arrival Time		2.860		

Figure 48 - Shortest Delay of Braun's Multiplier on VHDL circuits through paths

Power		Summary On-Chip
Total On-Chip Power:	0.086 W	
Junction Temperature:	25.4 °C	
Thermal Margin:	74.6 °C (16.2 W)	
Effective θ_{JA} :	4.6 °C/W	
Power supplied to off-chip devices:	0 W	
Confidence level:	Low	
Implemented Power Report		

Figure 50 - Braun's Multiplier Power Results from Vivado

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 19.433 ns	Worst Hold Slack (WHS): 1.860 ns	Worst Pulse Width Slack (WPWS): NA
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): NA
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: NA
Total Number of Endpoints: 16	Total Number of Endpoints: 16	Total Number of Endpoints: NA

All user specified timing constraints are met.

Figure 51 – Braun's Multiplier Timing Summary From Vivado

Figure 51 shows the results of the power given once the device has been synthesized on to the FPGA. Figure 52 shows the Design timing summary of Braun's multiplier.

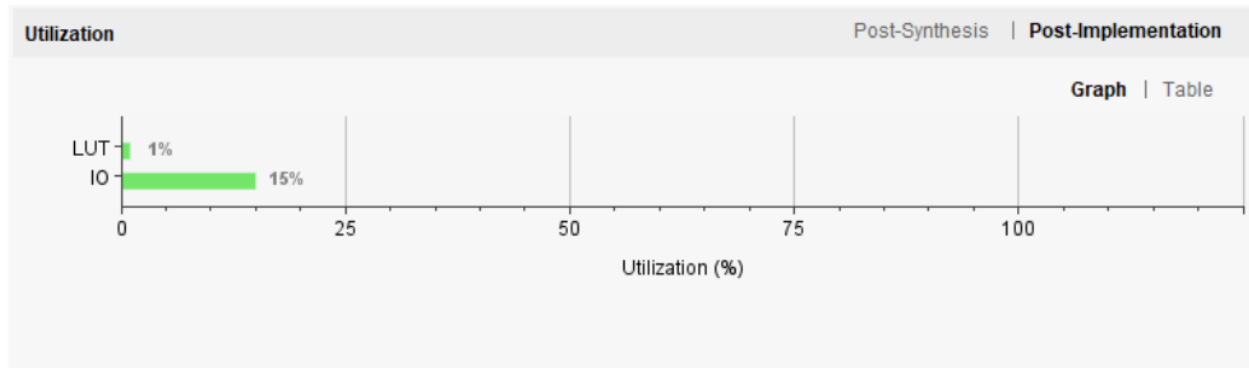


Figure 52 - Braun's Multiplier Chip utilization of FPGA from Vivado

Utilization				Post-Synthesis Post-Implementation
				Graph Table
Resource	Utilization	Available	Utilization %	
LUT	89	63400	0.14	
IO	32	210	15.24	

Figure 53 - Braun's multiplier chip utilisation of FPGA from Vivado

Figures 53 and 54 show the Utilization of Braun's multiplier of the FPGA once it has been synthesized.

8.2 Wallace Tree

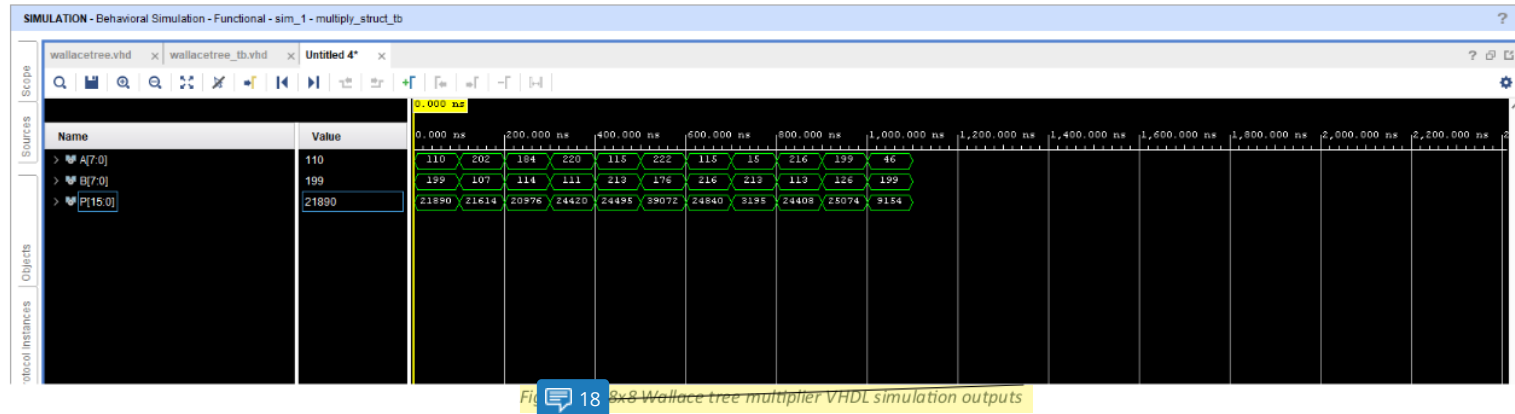


Fig. 18 32x8 Wallace tree multiplier VHDL simulation outputs

A test bench was created within Xilinx and 11 sample equations were made to test the output of the VHDL to ensure it's working correctly. Each of the 11 samples can be seen above in the unsigned decimal form. All the equations gave the right output therefore the code had been written correctly. Input A multiplied with input B gave the output P, the equations of the test bench can be found in appendix 2.

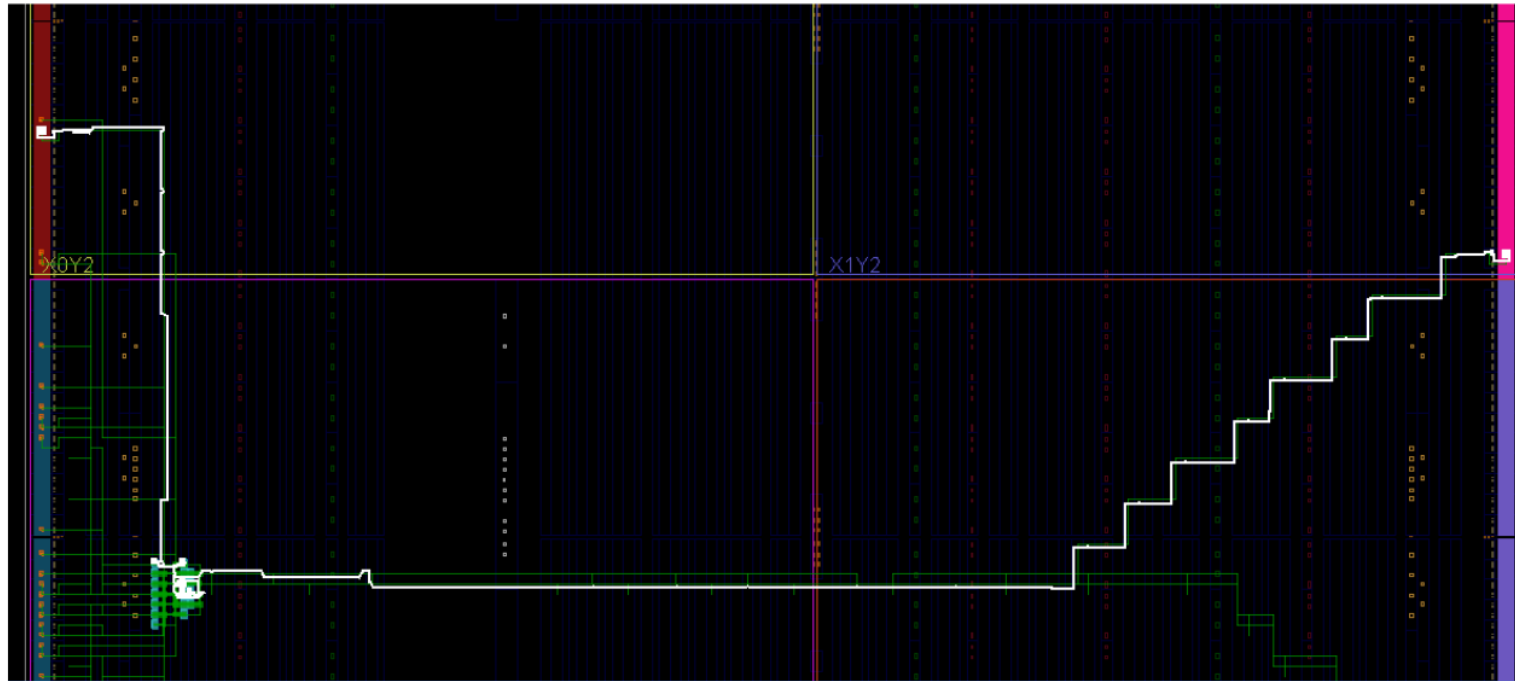


Figure 55 - Wallace tree multiplier longest delayed path synthesized onto FPGA circuit

Figures 56 and 57 show the FPGA circuitry which is due to the VHDL code having been synthesised onto it. The images show the maximum delay path in figure 56 and the minimum delay path in figure 57. Figures 58 and 59 show the minimum and maximum delay paths on the schematic of the VHDL Wallace tree circuit.

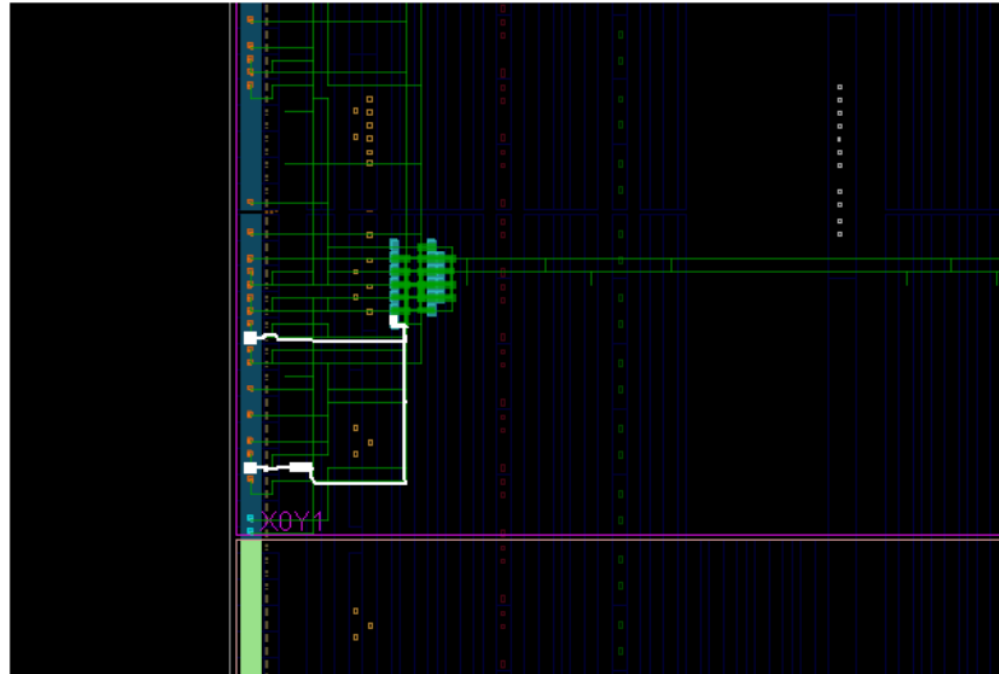


Figure 56 - Wallace tree multiplier shortest delay path on FPGA circuit

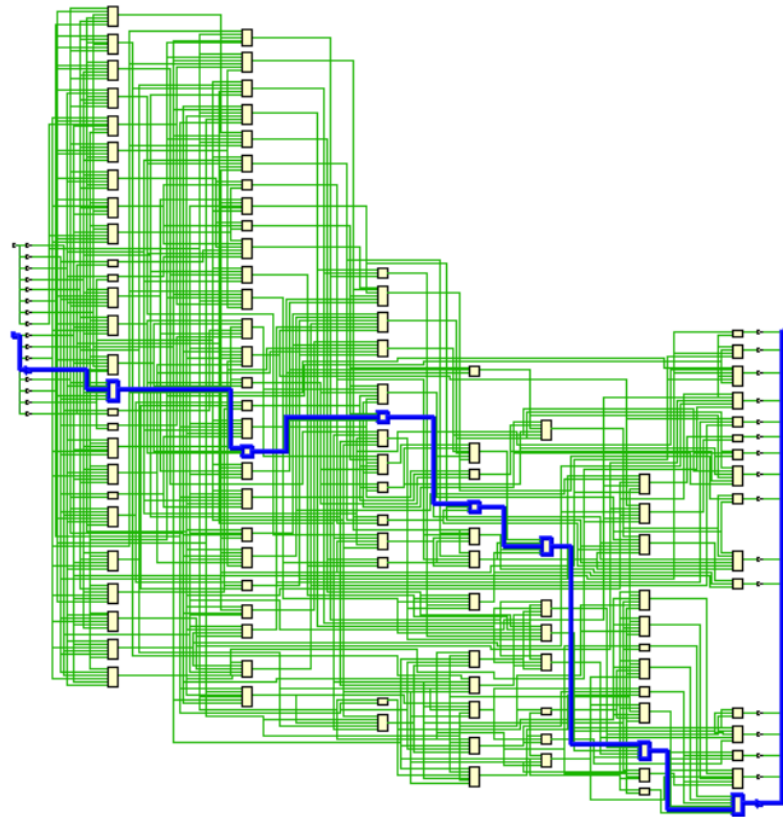


Figure 57 - Wallace Tree Multiplier Longest Delay path of circuit on the schematic

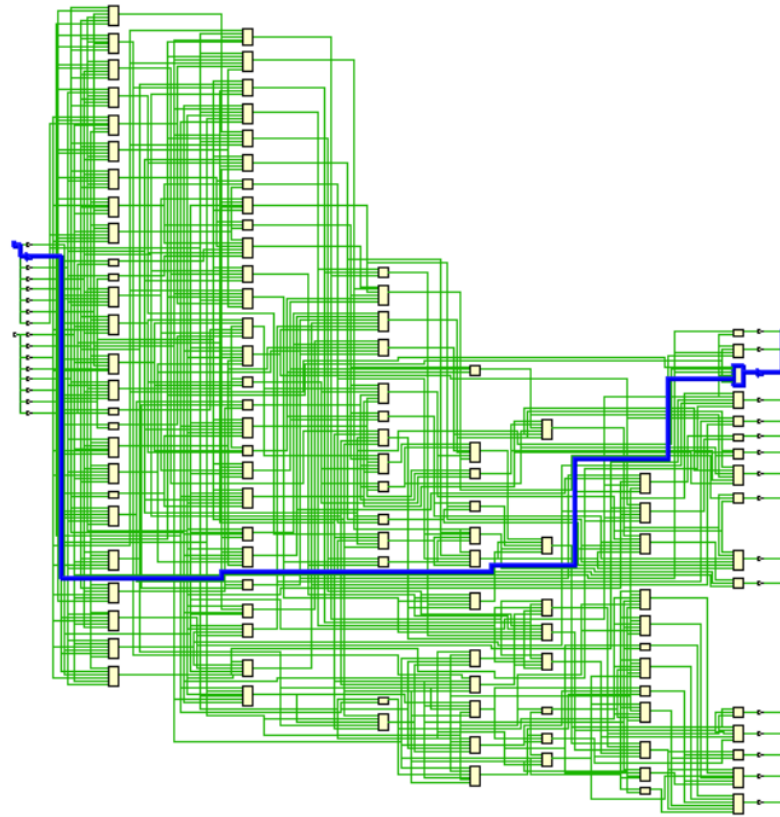


Figure 58 - Wallace Tree shortest delay of the circuit on schematic

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 1	13.107	9	22	B[3]	P[15]	17.893	5.572	12.321	31.000	input port clock		MaxDelay Path 31.000ns	0.000
Path 2	13.317	9	22	B[3]	P[13]	17.683	5.604	12.079	31.000	input port clock		MaxDelay Path 31.000ns	0.000
Path 3	13.948	9	22	B[3]	P[14]	17.052	5.587	11.465	31.000	input port clock		MaxDelay Path 31.000ns	0.000
Path 4	15.083	8	22	B[3]	P[11]	15.917	5.233	10.684	31.000	input port clock		MaxDelay Path 31.000ns	0.000
Path 5	15.100	8	22	B[3]	P[12]	15.900	5.232	10.668	31.000	input port clock		MaxDelay Path 31.000ns	0.000
Path 6	15.578	8	22	B[3]	P[10]	15.422	5.229	10.193	31.000	input port clock		MaxDelay Path 31.000ns	0.000
Path 7	16.306	6	22	B[3]	P[9]	14.694	4.940	9.754	31.000	input port clock		MaxDelay Path 31.000ns	0.000

Figure 59 - Wallace tree multiplier longest delay paths of the circuit

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception	Clock Uncertainty
Path 11	2.062	3	21	A[1]	P[2]	3.062	1.778	1.284	1.000	input port clock		MinDelay Path 1.000ns	0.000
Path 12	2.137	4	21	A[1]	P[3]	3.137	1.838	1.299	1.000	input port clock		MinDelay Path 1.000ns	0.000
Path 13	2.273	3	18	B[1]	P[1]	3.273	1.897	1.376	1.000	input port clock		MinDelay Path 1.000ns	0.000
Path 14	2.349	4	16	B[5]	P[7]	3.349	1.822	1.527	1.000	input port clock		MinDelay Path 1.000ns	0.000
Path 15	2.434	5	18	B[1]	P[4]	3.434	1.895	1.539	1.000	input port clock		MinDelay Path 1.000ns	0.000
Path 16	2.451	4	21	A[1]	P[9]	3.451	1.921	1.530	1.000	input port clock		MinDelay Path 1.000ns	0.000
Path 17	2.509	4	21	A[1]	P[5]	3.509	1.839	1.671	1.000	input port clock		MinDelay Path 1.000ns	0.000

Figure 60 - Wallace tree multiplier shortest delay paths of the circuit

In the figure's 60 and 61 above the shortest and longest delays of the Wallace tree are shown with the values of the delays.

Summary	
Name	Path 11
Slack (Hold)	2.062ns
Source	A[1] (input port)
Destination	P[2]
Path Group	**default**
Path Type	Min at Fast Process Corner
Requirement	1.000ns (MinDelay Path 1.000ns)
Data Path Delay	3.062ns (logic 1.778ns (58.074%) route 1.284ns (41.926%))
Logic Levels	3 (IBUF=1 LUT6=1 OBUF=1)
Output Delay	0.000ns
Timing ...ception	MinDelay Path 1.000ns

Figure 62 - Wallace tree multiplier's fastest delay of the VHDL circuit

Data Path				
Delay Type	Incr (ns)	Path ...	Location	Netlist Resource(s)
	(r) 0.000	0.000	Site: U18	A[1]
net (fo=0)	0.000	0.000		A[1]
			Site: U18	A_IBUF[1]_inst/I
IBUF (Prop_ibuf I_O)	(r) 0.441	0.441	Site: U18	A_IBUF[1]_inst/O
net (fo=21, routed)	0.499	0.940		A_IBUF[1]
			Site: S...X0Y66	P_OBUF[2]_inst_i_1/4
LUT6 (Prop_l6 I4_O)	(r) 0.062	1.002	Site: S...X0Y66	P_OBUF[2]_inst_i_1/O
net (fo=1, routed)	0.785	1.787		P_OBUF[2]
			Site: V14	P_OBUF[2]_inst/I
OBUF (Pr_buf I_O)	(r) 1.276	3.062	Site: V14	P_OBUF[2]_inst/O
net (fo=0)	0.000	3.062		P[2]
			Site: V14	P[2]

Figure 61 - Wallace tree fastest delay's route from input to output

Figures 63 and 62 show the shortest delay path that is available on the VHDL logic circuit. The time of the shortest delay is shown in figure 63 and the route of the shortest delay is shown in 62. Figures 64 to 66 show the longest delay path that is available on the VHDL logic circuit for a Wallace tree multiplier. Figure 65 shows the longest delay time and Figures 64 and 66 show the longest delay route from input to output.

Summary	
Name	Path 1
Slack	13.107ns
Source	B[3] (input port)
Destination	P[15]
Path Group	**default**
Path Type	Max at Slow Process Corner
Requirement	31.000ns (MaxDelay Path 31.000ns)
Data Path Delay	17.893ns (logic 5.572ns (31.139%) route 12.321ns (68.861%))
Logic Levels	9 (IBUF=1 LUT3=3 LUT5=2 LUT6=2 OBUF=1)
Output Delay	0.000ns
Timing ...ception	MaxDelay Path 31.000ns

Figure 64 - Wallace tree multiplier longest delay value

Data Path				
Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
	(r) 0.000	0.000	Site: H6	B[3]
net (fo=0)	0.000	0.000		B[3]
			Site: H6	B_IBUF[3]_instI
IBUF (Prop_ibuf_1_O)	(r) 1.003	1.003	Site: H6	B_IBUF[3]_instI/O
net (fo=22, routed)	4.938	5.940		B_IBUF[3]
			Site: S...X2Y70	P_OBUF[9]_inst_i_18/I2
LUT6 (Pro...t6_I2_O)	(r) 0.148	6.088	Site: S...X2Y70	P_OBUF[9]_inst_i_18/O
net (fo=3, routed)	0.925	7.014		P_OBUF[9]_inst_i_18_n_0
			Site: S...X2Y69	P_OBUF[9]_inst_i_13/I0
LUT3 (Pro...t3_I0_O)	(f) 0.157	7.171	Site: S...X2Y69	P_OBUF[9]_inst_i_13/O
net (fo=3, routed)	0.636	7.807		P_OBUF[9]_inst_i_13_n_0
			Site: S...X3Y69	P_OBUF[9]_inst_i_4/I1
LUT3 (Pro...t3_I1_O)	(f) 0.393	8.200	Site: S...X3Y69	P_OBUF[9]_inst_i_4/O
net (fo=3, routed)	0.633	8.833		P_OBUF[9]_inst_i_4_n_0
			Site: S...X5Y69	P_OBUF[12]_inst_i_8/I0
LUT3 (Pro...t3_I0_O)	(f) 0.362	9.195	Site: S...X5Y69	P_OBUF[12]_inst_i_8/O
net (fo=1, routed)	0.762	9.956		P_OBUF[12]_inst_i_8_n_0
			Site: S...X3Y70	P_OBUF[12]_inst_i_2/I2

Figure 63 - Wallace tree multiplier longest delay path route from input to output

LUT5 (Pro...t5_12_O)	(r) 0.148	10.104	Site: S...X3Y70	◀ P_OBUF[12]_inst_i_2/O
net (fo=3, routed)	0.514	10.619		↗ P_OBUF[12]_inst_i_2_n_0
			Site: S...X3Y72	▶ P_OBUF[15]_inst_i_6/I0
LUT5 (Pro...t5_10_O)	(r) 0.156	10.775	Site: S...X3Y72	◀ P_OBUF[15]_inst_i_6/O
net (fo=3, routed)	0.896	11.671		↗ P_OBUF[15]_inst_i_6_n_0
			Site: S...X1Y72	▶ P_OBUF[15]_inst_i_1/I4
LUT6 (Pro...t6_14_O)	(r) 0.362	12.033	Site: S...X1Y72	◀ P_OBUF[15]_inst_i_1/O
net (fo=1, routed)	3.017	15.050		↗ P_OBUF[15]
			Site: H17	▶ P_OBUF[15]_inst/I
OBUF (Pr...buf_1_O)	(r) 2.843	17.893	Site: H17	◀ P_OBUF[15]_inst/O
net (fo=0)	0.000	17.893		↗ P[15]
			Site: H17	◀ P[15]

Figure 65 - Wallace tree multiplier longest delay route from input to output continued

Power		Summary On-Chip
Total On-Chip Power:	0.086 W	
Junction Temperature:	25.4 °C	
Thermal Margin:	74.6 °C (16.2 W)	
Effective θ_{JA} :	4.6 °C/W	
Power supplied to off-chip devices:	0 W	
Confidence level:	Low	
Implemented Power Report		

Figure 66 - Wallace Tree Multiplier Power results after the logic circuit was synthesised onto FPGA

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 13.107 ns	Worst Hold Slack (WHS): 2.062 ns	Worst Pulse Width Slack (WPWS): NA
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): NA
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: NA
Total Number of Endpoints: 16	Total Number of Endpoints: 16	Total Number of Endpoints: NA
All user specified timing constraints are met.		

Figure 67 - Wallace tree multiplier Design timing summary results from timings applied to FPGA

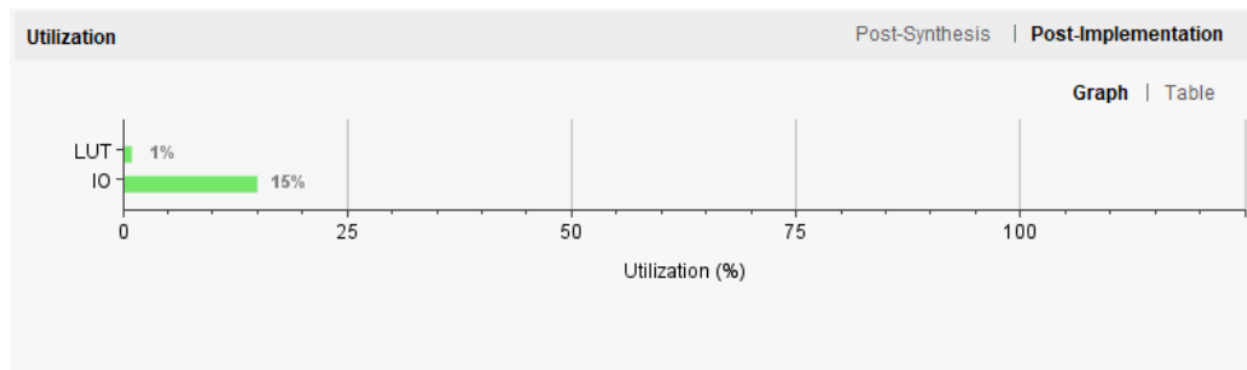


Figure 68 - Wallace tree multiplier utilization of FPGA after the VHDL circuit had been synthesised onto it

Utilization				Post-Synthesis	Post-Implementation
				Graph Table	
Resource	Utilization	Available	Utilization %		
LUT	89	63400	0.14		
IO	32	210	15.24		

Figure 69 - Wallace tree multiplier utilization results after VHDL circuit had been synthesised onto it

Figure 69 and 70 show the utilization of the FPGA by the Wallace tree multiplier. Figure 68 shows the design timing summary of the Wallace tree multiplier within Vivado from the VHDL logic circuit being applied to the FPGA. Figure 67 shows the power results from the multiplier being applied onto the FPGA.

9. Details of testing and results

With the use of Vivado, each of the multipliers could be tested and could generate the results to compare one against the other. These results included the on-chip power consumption of each of the multipliers, The utilization of both multipliers on the FPGA and also the timing constraints of each of the multipliers.

For each of the multipliers, they were given the same sampling multiplication equations on the test bench. This was to keep a consistency between the two multipliers for comparison between them. These are shown in figures 38 and 55.

By using the logic circuit designed for the 8x8 Braun's multiplier, the maximum and minimum delay was calculated. Also the minimum and maximum delay value was calculated and implemented for the Wallace tree multiplier. Using these values and implementing them onto Vivado allowed us to get a design timing summary of the multiplier with the VHDL code written which was shown in figure 52 for Braun's multiplier and figure 68 for the Wallace tree multiplier.

The Power results and the utilization of the FPGA of the VHDL written for Braun's multiplier were given in figure 51 and for Wallace tree multiplier were given in figure 67. The results were given after the code was synthesised onto FPGA.

9.1 8x8 Braun's multiplier and 8x8 Wallace tree multiplier results

	8x8 Braun's Multiplier	8x8 Wallace Tree Multiplier
Total On-Chip Power:	0.086W	0.086W
Junction Temperature:	25.4 C	25.4 C
Thermal Margin:	74.6 C	74.6 C
Worst Negative Slack:	13.107nS	19.433nS
Worst Hold Slack:	2.062nS	1.860nS

Table 1 – Braun's Multiplier and Wallace Tree multiplier comparison table

10. Comparisons between the multipliers

Results were gathered from both multipliers. Using these results, both multipliers could be compared against one another, comparing their minimum and maximum delay as well as their power consumption on the chip. The goal is for the multiplier to have the fastest speeds possible and the lowest power consumption and chip utilization.

When looking at figure's 51 and 67 both multipliers which have been written in structural VHDL code consume the same amount of power. This is unusual and the values should be different but for this scenario, both the power consumption and the Junction terminals are the same.

When comparing the utilization of the FPGA chip, both multipliers use the same amount of I/O's and also the same amount of LUT on the Field programmable gate array. Due to this and the previous result on power consumption, both the utilization and the power consumption of each multiplier are the same.

When it comes down to comparing the timings of the multipliers. Each of the multipliers has the same minimum delay of 1nS. The Wallace tree multiplier has the fastest maximum delay of the two shown in figure.. which was 31nS. The Braun's multiplier had a maximum delay of 41nS. Therefore from the logic circuit, the largest delay was Braun's multiplier. The delays were placed onto the multipliers within Vivado as the timing constraints.

The timing constraints gave back a set of results which were the minimum and maximum paths of the multipliers. The slack of each of the multipliers was given. Braun's multiplier slack is given in figure 52 and Wallace tree multiplier slack is given in figure 68. Both multipliers met the timing requirements and therefore have a positive slack. The Wallace tree multiplier had a greater WNS (worse negative slack) than Braun's multiplier. Wallace tree with a WNS of 13.107nS and the Braun's multi with a WNS of 19.433nS. For the shortest slack WHS (worst hold slack) the Braun's multiplier was slightly greater than the Wallace tree with Braun's being 1.860nS and Wallace tree being 2.062nS. The slack is simply the difference between the time the signal was expected to the actual arriving time of the signal.

The critical path of the Wallace tree multiplier is path 1 shown in figure 65. This is the path with the longest delay which was 17.893nS. The critical path of Braun's multiplier was 21.567nS which was also path 1. This gives Braun's multiplier a longer overall delay on the Critical Path of the FPGA.

The shortest path delay of the Wallace tree multiplier was Path 2 and 2.062nS shown in figure 46 The shortest path delay of Braun's multiplier was Path 11 and 1.860nS. This gave Braun's multiplier the shortest path delay of both the multipliers.

11. Conclusion

The conclusion from results gathered from simulating each of the multipliers concluded that the fastest of the two was the Wallace tree multiplier. The timings of the Wallace tree proved that the multiplier was faster in the builds that I built them. This is due to fewer adders needing to be travelled through from inputs to outputs. The Wallace tree featured 48 Full adders, 16 Half adders and 64 AND gates. The Braun's multiplier features 48 Full adders, 8 Half adders and 64 AND gates. This gave the Braun's Multiplier fewer gates to compute but the path from input-output on most paths required more gates to be travelled through on the Braun's Multiplier. The speed and timing of the multipliers can be improved by increasing the multiplier clock times. The comparisons of the power consumption cannot be made with these simulations due to the same values being given for each of the multipliers.

The code for the Braun's multiplier can be seen in Appendix 2 and the code for the Wallace tree multiplier can be seen in Appendix 1.

12. Appendices

12.1 Appendix 1

12.1.1 Wallace Tree VHDL Code

```
wallacetreec.vhd x wallacetreec_tb.vhd x Untitled 4* x
C:/Users/s2116739/Documents/Semester 2/LW 6/Electronics 2/project_wallacetreec/project_wallacetreec/srcs/sources_1/new/wallacetreec.vhd

21 library ieee;
22 use ieee.std_logic_1164.all;
23
24
25 entity AND2 is -- AND Entity
26 port(
27     A,B: in BIT;
28     Z : buffer BIT);
29 end AND2;
30
31 Architecture Behavioural of AND2 is -- AND architecture
32 begin
33     Z <= A and B;
34 end Behavioural;
35
36 entity half_adder is -- Half Adder Entity
37 port (a, b : in BIT;
38     sum, carry : buffer BIT
39 );
40 end half_adder;
41
42 Architecture Behavioural of half_adder is -- Half Adder architecture
43 begin
44     sum <= a xor b;
45     carry <= a and b;
46 end Behavioural;
47
48 entity full_adder is -- Full Adder Entity
49 port (a, b, c : in BIT;
50     sum, carry : buffer BIT
51 );
52 end full_adder;
53
54 Architecture Behavioural of full_adder is -- Full Adder architecture
55 begin
56     sum <= a xor b xor c;
57     carry <= (a and b) or (a and c) or (b and c);
58 end Behavioural;
59
60 entity multiply_struct is
61 port (A, B : in bit_vector(7 downto 0);
62     P : buffer bit_vector(15 downto 0)
63 );
64 end multiply_struct;
```

wallacetre.vhd × wallacetre_tb.vhd × Untitled 4* ×

C:/Users/s2116739/Documents/Semester 2 Lvl 6/Electronics 2/project_wallacetre/project_wallacetre.srscs/sources_1/new/wallacetre.vhd

Q

🏠

↩

➡

🔍

📄

📁

✂

🔗

🔧

🔌

🔥

⏏

🔍

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

```
architecture structural of multiply_struct is
    component AND2
    port (A, B : in BIT;
        X : buffer BIT);
    end component;

    component half_adder
    port (A, B : in BIT;
        sum, carry : buffer BIT);
    end component;

    component full_adder
    port (a, b, c : in BIT;
        sum, carry : buffer BIT
    );
    end component;

    signal S: bit_vector (112 downto 1);
    signal C: bit_vector (64 downto 1);

    begin

        A1: AND2 port map(A(0),B(0),P(0)); -- stage 0
        A2: AND2 port map(A(1),B(0),S(1));
        A3: AND2 port map(A(2),B(0),S(52));
        A4: AND2 port map(A(3),B(0),S(53));
        A5: AND2 port map(A(4),B(0),S(54));
        A6: AND2 port map(A(5),B(0),S(55));
        A7: AND2 port map(A(6),B(0),S(56));
        A8: AND2 port map(A(7),B(0),S(57));

        A9: AND2 port map(A(0),B(1),S(58));
        A10: AND2 port map(A(1),B(1),S(59));
        A11: AND2 port map(A(2),B(1),S(60));
        A12: AND2 port map(A(3),B(1),S(61));
        A13: AND2 port map(A(4),B(1),S(62));
        A14: AND2 port map(A(5),B(1),S(63));
        A15: AND2 port map(A(6),B(1),S(64));
        A16: AND2 port map(A(7),B(1),S(65));
```

59

Sources Scope
 Protocol Instances Objects

wallacetree.vhd x wallacetree_tb.vhd x Untitled 4* x

C:\Users\sis116739\Documents\Semester 2\Lv 6\Electronics 2\project_wallacetree\project_wallacetree.srscs/sources_1new\wallacetree.vhd

```

106: A17: AND2 port map (A(0), B(2), S(66));
107: A18: AND2 port map (A(1), B(2), S(67));
108: A19: AND2 port map (A(2), B(2), S(68));
109: A20: AND2 port map (A(3), B(2), S(69));
110: A21: AND2 port map (A(4), B(2), S(70));
111: A22: AND2 port map (A(5), B(2), S(71));
112: A23: AND2 port map (A(6), B(2), S(72));
113: A24: AND2 port map (A(7), B(2), S(73));
114:
115: A25: AND2 port map (A(0), B(3), S(91));
116: A26: AND2 port map (A(1), B(3), S(94));
117: A27: AND2 port map (A(2), B(3), S(95));
118: A28: AND2 port map (A(3), B(3), S(96));
119: A29: AND2 port map (A(4), B(3), S(97));
120: A30: AND2 port map (A(5), B(3), S(98));
121: A31: AND2 port map (A(6), B(3), S(99));
122: A32: AND2 port map (A(7), B(3), S(100));
123:
124: A33: AND2 port map (A(0), B(4), S(81));
125: A34: AND2 port map (A(1), B(4), S(82));
126: A35: AND2 port map (A(2), B(4), S(83));
127: A36: AND2 port map (A(3), B(4), S(84));
128: A37: AND2 port map (A(4), B(4), S(85));
129: A38: AND2 port map (A(5), B(4), S(86));
130: A39: AND2 port map (A(6), B(4), S(87));
131: A40: AND2 port map (A(7), B(4), S(88));
132:
133: A41: AND2 port map (A(0), B(5), S(69));
134: A42: AND2 port map (A(1), B(5), S(90));
135: A43: AND2 port map (A(2), B(5), S(91));
136: A44: AND2 port map (A(3), B(5), S(92));
137: A45: AND2 port map (A(4), B(5), S(93));
138: A46: AND2 port map (A(5), B(5), S(94));
139: A47: AND2 port map (A(6), B(5), S(95));
140: A48: AND2 port map (A(7), B(5), S(96));
141:
142: A49: AND2 port map (A(0), B(6), S(97));
143: A50: AND2 port map (A(1), B(6), S(98));
144: A51: AND2 port map (A(2), B(6), S(99));
145: A52: AND2 port map (A(3), B(6), S(100));
146: A53: AND2 port map (A(4), B(6), S(101));
147: A54: AND2 port map (A(5), B(6), S(102));
148: A55: AND2 port map (A(6), B(6), S(103));
149: A56: AND2 port map (A(7), B(6), S(104));
150:
  
```


12.2 Appendix 2

12.2.1 Braun's Multiplier VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;

entity AND2 is -- AND Entity
port(
  A,B: in BIT;
  x : buffer BIT);
end AND2;

architecture behaviour of AND2 is -- AND architecture
begin
  x <= A and B;
end behaviour;

entity half_adder is -- Half Adder Entity
port (a, b : in BIT;
      sum, carry : buffer BIT
    );
end half_adder;

architecture behaviour of half_adder is -- Half Adder architecture
begin
  sum <= a xor b;
  carry <= a and b;
end behaviour;

entity full_adder is -- Full Adder Entity
port (a, b, c : in BIT;
      sum, carry : buffer BIT
    );
end full_adder;

architecture behaviour of full_adder is -- Full Adder architecture
begin
  sum <= a xor b xor c;
  carry <= (a and b) or (a and c) or (b and c);
end behaviour;

entity multiply_struct is
port (A, B : in bit_vector(7 downto 0);
      P : buffer bit_vector(15 downto 0)
    );
end multiply_struct;
```

architecture structural of multiply_struct is

```
component AND2
port (A, B : in BIT;
X : buffer BIT);
end component;

component half_adder
port (A, B : in BIT;
sum, carry : buffer BIT);
end component;

component full_adder
port (a, b, c : in BIT;
sum, carry : buffer BIT
);
end component;
signal S: bit_vector (105 downto 1);
Signal C: bit_vector (55 downto 1);

begin

A1: AND2 port map(A(0),B(0),P(0)); -- top line
A2: AND2 port map(A(1),B(0),S(1));
A3: AND2 port map(A(2),B(0),S(2));
A4: AND2 port map(A(3),B(0),S(3));
A5: AND2 port map(A(4),B(0),S(4));
A6: AND2 port map(A(5),B(0),S(5));
A7: AND2 port map(A(6),B(0),S(6));
A8: AND2 port map(A(7),B(0),S(7));

A9: AND2 port map(A(0),B(1),S(8)); -- 2nd line
A10: AND2 port map(A(1),B(1),S(9));
A11: AND2 port map(A(2),B(1),S(10));
A12: AND2 port map(A(3),B(1),S(11));
A13: AND2 port map(A(4),B(1),S(12));
A14: AND2 port map(A(5),B(1),S(13));
A15: AND2 port map(A(6),B(1),S(14));
A16: AND2 port map(A(7),B(1),S(15));

HA1: half_adder port map(S(8),S(1),P(1),C(1));
HA2: half_adder port map(S(15),C(7),S(70),C(8));

FA1: full_adder port map(S(9),S(2),C(1),S(64),C(2));
FA2: full_adder port map(S(10),S(3),C(2),S(65),C(3));
FA3: full_adder port map(S(11),S(4),C(3),S(66),C(4));
```



```
FA4: full_adder port map(S(12),S(5),C(4),S(67),C(5));  
FA5: full_adder port map(S(13),S(6),C(5),S(68),C(6));  
FA6: full_adder port map(S(14),S(7),C(6),S(69),C(7));
```

```
A17: AND2 port map(A(0),B(2),S(16)); -- 3rd line  
A18: AND2 port map(A(1),B(2),S(17));  
A19: AND2 port map(A(2),B(2),S(18));  
A20: AND2 port map(A(3),B(2),S(19));  
A21: AND2 port map(A(4),B(2),S(20));  
A22: AND2 port map(A(5),B(2),S(21));  
A23: AND2 port map(A(6),B(2),S(22));  
A24: AND2 port map(A(7),B(2),S(23));
```

```
HA3: half_adder port map(S(64),S(16),P(2),C(9));
```

```
FA7: full_adder port map(S(17),S(65),C(9),S(71),C(10));  
FA8: full_adder port map(S(18),S(66),C(10),S(72),C(11));  
FA9: full_adder port map(S(19),S(67),C(11),S(73),C(12));  
FA10: full_adder port map(S(20),S(68),C(12),S(74),C(13));  
FA11: full_adder port map(S(21),S(69),C(13),S(75),C(14));  
FA12: full_adder port map(S(22),S(70),C(14),S(76),C(15));  
FA13: full_adder port map(S(23),C(8),C(15),S(77),C(16));
```

```
A25: AND2 port map(A(0),B(3),S(24)); -- 4th line  
A26: AND2 port map(A(1),B(3),S(25));  
A27: AND2 port map(A(2),B(3),S(26));  
A28: AND2 port map(A(3),B(3),S(27));  
A29: AND2 port map(A(4),B(3),S(28));  
A30: AND2 port map(A(5),B(3),S(29));  
A31: AND2 port map(A(6),B(3),S(30));  
A32: AND2 port map(A(7),B(3),S(31));
```

```
HA4: half_adder port map(S(71),S(24),P(3),C(55));
```

```
FA14: full_adder port map(S(25),S(72),C(55),S(78),C(17));  
FA15: full_adder port map(S(26),S(73),C(17),S(79),C(18));  
FA16: full_adder port map(S(27),S(74),C(18),S(80),C(19));  
FA17: full_adder port map(S(28),S(75),C(19),S(81),C(20));  
FA18: full_adder port map(S(29),S(76),C(20),S(82),C(21));  
FA19: full_adder port map(S(30),S(77),C(21),S(83),C(22));  
FA20: full_adder port map(S(31),C(16),C(22),S(84),C(23));
```

```
A33: AND2 port map(A(0),B(4),S(32)); -- 5th line  
A34: AND2 port map(A(1),B(4),S(33));  
A35: AND2 port map(A(2),B(4),S(34));  
A36: AND2 port map(A(3),B(4),S(35));  
A37: AND2 port map(A(4),B(4),S(36));
```

```

A38: AND2 port map(A(5),B(4),S(37));
A39: AND2 port map(A(6),B(4),S(38));
A40: AND2 port map(A(7),B(4),S(39));

HA5: half_adder port map(S(78),S(32),P(4),C(24));

FA21: full_adder port map(S(33),S(79),C(24),S(85),C(25));
FA22: full_adder port map(S(34),S(80),C(25),S(86),C(26));
FA23: full_adder port map(S(35),S(81),C(26),S(87),C(27));
FA24: full_adder port map(S(36),S(82),C(27),S(88),C(28));
FA25: full_adder port map(S(37),S(83),C(28),S(89),C(29));
FA26: full_adder port map(S(38),S(84),C(29),S(90),C(30));
FA27: full_adder port map(S(39),C(23),C(30),S(91),C(31));

A41: AND2 port map(A(0),B(5),S(40)); -- 6th line
A42: AND2 port map(A(1),B(5),S(41));
A43: AND2 port map(A(2),B(5),S(42));
A44: AND2 port map(A(3),B(5),S(43));
A45: AND2 port map(A(4),B(5),S(44));
A46: AND2 port map(A(5),B(5),S(45));
A47: AND2 port map(A(6),B(5),S(46));
A48: AND2 port map(A(7),B(5),S(47));

HA6: half_adder port map(S(85),S(40),P(5),C(32));

FA28: full_adder port map(S(41),S(86),C(32),S(92),C(33));
FA29: full_adder port map(S(42),S(87),C(33),S(93),C(34));
FA30: full_adder port map(S(43),S(88),C(34),S(94),C(35));
FA31: full_adder port map(S(44),S(89),C(35),S(95),C(36));
FA32: full_adder port map(S(45),S(90),C(36),S(96),C(37));
FA33: full_adder port map(S(46),S(91),C(37),S(97),C(38));
FA34: full_adder port map(S(47),C(31),C(38),S(98),C(39));

A49: AND2 port map(A(0),B(6),S(48)); -- 7th line
A50: AND2 port map(A(1),B(6),S(49));
A51: AND2 port map(A(2),B(6),S(50));
A52: AND2 port map(A(3),B(6),S(51));
A53: AND2 port map(A(4),B(6),S(52));
A54: AND2 port map(A(5),B(6),S(53));
A55: AND2 port map(A(6),B(6),S(54));
A56: AND2 port map(A(7),B(6),S(55));

HA7: half_adder port map(S(92),S(48),P(6),C(40));

FA35: full_adder port map(S(49),S(93),C(40),S(99),C(41));
FA36: full_adder port map(S(50),S(94),C(41),S(100),C(42));
FA37: full_adder port map(S(51),S(95),C(42),S(101),C(43));

```

```
FA38: full_adder port map(S(52),S(96),C(43),S(102),C(44));
FA39: full_adder port map(S(53),S(97),C(44),S(103),C(45));
FA40: full_adder port map(S(54),S(98),C(45),S(104),C(46));
FA41: full_adder port map(S(55),C(39),C(46),S(105),C(47));
```

```
A57: AND2 port map(A(0),B(7),S(56)); -- 8th line
```

```
A58: AND2 port map(A(1),B(7),S(57));
```

```
A59: AND2 port map(A(2),B(7),S(58));
```

```
A60: AND2 port map(A(3),B(7),S(59));
```

```
A61: AND2 port map(A(4),B(7),S(60));
```

```
A62: AND2 port map(A(5),B(7),S(61));
```

```
A63: AND2 port map(A(6),B(7),S(62));
```

```
A64: AND2 port map(A(7),B(7),S(63));
```

```
HA8: half_adder port map(S(99),S(56),P(7),C(48));
```

```
FA42: full_adder port map(S(57),S(100),C(48),P(8),C(49));
```

```
FA43: full_adder port map(S(58),S(101),C(49),P(9),C(50));
```

```
FA44: full_adder port map(S(59),S(102),C(50),P(10),C(51));
```

```
FA45: full_adder port map(S(60),S(103),C(51),P(11),C(52));
```

```
FA46: full_adder port map(S(61),S(104),C(52),P(12),C(53));
```

```
FA47: full_adder port map(S(62),S(105),C(53),P(13),C(54));
```

```
FA48: full_adder port map(S(63),C(47),C(54),P(14),P(15));
```

```
end structural;
```

12.2.2 Braun's Multiplier Test Bench

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Numeric_Std.all;

entity multiply_struct_tb is
end;

architecture bench of multiply_struct_tb is

    component multiply_struct
    port (A, B : in bit_vector(7 downto 0);
          P : buffer bit_vector(15 downto 0)
        );
    end component;

    signal A, B: bit_vector(7 downto 0);
    signal P: bit_vector(15 downto 0) ;

begin

    uut: multiply_struct port map ( A => A,
                                   B => B,
                                   P => P );

    stimulus: process
    constant period: time := 100ns;
    begin

        A <= "01101110";
        B <= "11000111";
        wait for period;

        A <= "11001010";
        B <= "01101011";
        wait for period;

        A <= "10111000";
        B <= "01110010";
        wait for period;

        A <= "11011100";
        B <= "01101111";
        wait for period;

        A <= "01110011";
```

```
B <= "11010101";  
wait for period;
```

```
A <= "11011110";  
B <= "10110000";  
wait for period;
```

```
A <= "01110011";  
B <= "11011000";  
wait for period;
```

```
A <= "00001111";  
B <= "11010101";  
wait for period;
```

```
A <= "11011000";  
B <= "01110001";  
wait for period;
```

```
A <= "11000111";  
B <= "01111110";  
wait for period;
```

```
A <= "00101110";  
B <= "11000111";  
wait for period;
```

```
wait;  
end process;
```

```
end;
```

13. References

- [1] S. K. Soniya, "A Review of Different Type of Multipliers and Multiplier-Accumulator Unit.," *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, vol. 2, no. 4, pp. 364-368, 2013.
- [2] P. J. Ashenden, *Digital Design (VHDL): An Embedded Systems Approach Using VHDL*, Morgan Kaufmann, 2007.
- [3] S. K. Soniya1, "A Review of Different Type of Multipliers and Multiplier-Accumulator Unit.," *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, vol. 2, no. 4, p. 364, 2013.
- [4] P. P. G. B. S. Dr. Ravi Shankar Mishra, "Design and Implements of Booth and Robertson's multipliers algorithm on FPGA," *International Journal of Engineering Research and Applications (IJERA)*, vol. 1, no. 3, pp. 905-910.
- [5] D. S. ROY, "Wallace and Dadda Multiplier Design," 18 06 2019. [Online]. Available: <https://digitalsystemdesign.in/wallace-and-dadda-multiplier-design/>.
- [6] "Nexys A7 Reference Manual," Digilent, [Online]. Available: <https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual>. [Accessed 26 04 2023].
- [7] teamques10, "Draw the flowchart of Booth's Algorithm and multiply $(-7)*(3)$ using Booth's algorithm," Ques10, 04 22. [Online]. Available: <https://www.ques10.com/p/36454/draw-the-flowchart-of-booths-algorithm-and-multiply-1/>. [Accessed 15 05 23].

Comparison of Brauns Multiplier and Wallace tree multiplier using VHDL

GRADEMARK REPORT

FINAL GRADE

82/100

GENERAL COMMENTS

Instructor

a. Introduction to Multipliers/ Various Multipliers/ Advantages and Disadvantages/ Applications of Multipliers (8) : 6 Marks

Well said- The writing clearly shows that you have well understood the assignment criteria Benefits of multipliers were clearly Mentioned.

b. Background Research (8) : 5 Marks

Engagement with an extensive range of relevant literature is seen.

c. Diversity of sources and Referencing Skills(4) : 3 Marks

Consistently accurate application of referencing.

a. Task 1: Identifying the correct Multiplier (4), Architecture of Braun's Multiplier(6) : 8 Marks

Well Done! The correct Multiplier is identified. Its key concepts were explained. It is evident that you have a clear idea about multipliers. Architecture of 8X8 Braun's is well captured. Assumptions of partial signals made is very clear. More points are required in explaining Braun's Multiplier.

**b. Task 2: Dot Diagram(4),
Architecture of Wallace Multiplier(6) :
9 Marks**

Excellent Work- The Dot Diagram is well Captured. Extensive usage of Full Adders and Half adders were Presented. It is very clear the you have a good knowledge on choosing the correct adder in your architecture. The assumptions on Partial Signals are very clear.

**a. Finding
Critical Path, Longest Delay, Min and Max Delay, Power
Consumption(12). :9 Marks**

Good Job- Critical evaluation on finding the Critical path, longest min, max delay, Power consumption were made. The range slightly differs with varies respect to the no. of LUTs and I/O used. Based upon the Evidence Based judgements the solutions you have stated for delay are within the expected range. But the Power consumption for Wallace should be different. All the needed screen dumps were found which evidently proves the work of yours. It is an added credit that you have managed to compare the all the criteria's with both the Multipliers.

**b. Innovative
thinking in problem solving and feature additions(8) : 7 Marks**

Clearly distinguishes between fact, opinion, & acknowledges value judgments are seen.

a. Simulation (7) :7 Marks

Well done - The simulation results are well captured by using appropriate screen dumps. Minimum 4 samples were found. The usage of test bench were flawless.

**b. VHDL Coding's
(10) :9 Marks**

Good - Both the codes are relevant and produces the expected outputs and results without any errors. The resultant Products are exactly matching with the manual Calculations

b. Circuit Schematic (4) :4 Marks

Good- The screen dump provided for the Circuit schematic is neat and precise in both the cases of multipliers. It covers most of the Blocks.

d. FPGA Synthesis (4) :3 Marks

Synthesis is provided. It clearly shows that the you have a very good experience in obtaining the synthesis and Implementation reports. The procedures to capture FPGA synthesis were done perfectly

a. Logical order of ideas, Flow of ideas in the report from the beginning to end with appropriate chapters, sub chapters and continuation(7) :6 Marks

Work is exceptionally coherent, very fluent and is presented professionally.

b. Definitions, Acronyms and Abbreviations (3) :2 Marks

Excellent- Definitions, Acronyms and Abbreviations used appropriately and adequately.

c. Professionalism, Spelling and Grammar, Complete Report, Proper Presentation, Quality of the report(5) :4 Marks

Well Done- An precise level of professionalism is maintained throughout the report. Proper Presentation and the Quality of the report is flawless.

Total Mark :
82 Marks

Findings from the Code and Simulations

Codes have port and signal declarations . No errors or warnings found when compiling the code. More than 4 samples where simulated and the results are up to the expectations.

[Click here to view the Feed back Report](#)

PAGE 1

PAGE 2



Comment 1

Well Organized

PAGE 3



Comment 2

Advantages were captured.



Comment 3

True



Comment 4

Good- Identification, the first multiplier is Braun's



Comment 5

Good

PAGE 4

PAGE 5



Comment 6

You have made the research work very general and the depth of the Multipliers should cover be more to achieve good marks.



Comment 7

Well Said

PAGE 6



Comment 8

Good try , But there are many other products such as Sequential , combinational , Array Multipliers are seen in the market.

PAGE 7



Comment 9

Assumptions of partial signals made is very clear.

PAGE 8



Comment 10

Good

PAGE 9

PAGE 10

PAGE 11



Comment 11

More points are required in explaining Braun's Multiplier.

PAGE 12



Comment 12

Architecture of 8X8 Braun's is captured.

PAGE 13



Comment 13

Good Job- The Dot diagram is correct and well captured. The diagram is very clear and precise.



PAGE 14



Comment 14

There is a clear understanding in the usage of Full Adders and half Adders.

PAGE 15



Comment 15

Architecture of Wallace is also seen. The consideration of the Partial Signals are also found which is little clumsy.

PAGE 16
PAGE 17
PAGE 18
PAGE 19
PAGE 20
PAGE 21
PAGE 22
PAGE 23
PAGE 24
PAGE 25
PAGE 26
PAGE 27
PAGE 28
PAGE 29
PAGE 30
PAGE 31
PAGE 32
PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37



Comment 16

Satisfactory -Simulation Found for Braun's which is also verified.

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42



Comment 17

Good- Critical evaluation on finding the longest min, max delay were made. The delay found within the expected range.

PAGE 43

PAGE 44

PAGE 45

PAGE 46



Comment 18

Simulation Found for wallace.



Comment 19

Critical evaluation on finding the longest min, max delay were made. The delay found within the expected range.



Comment 20

The power consumption will not be same for both the multipliers.

PAGE 60

PAGE 61

PAGE 62

PAGE 63

PAGE 64

PAGE 65

PAGE 66

PAGE 67

PAGE 68



Comment 21

Findings from the Code and Simulations

Codes have port and signal declarations . No errors or warnings found when compiling the code. More than 4 samples where simulated and the results are up to the expectations

PAGE 69

PAGE 70

PAGE 71

PAGE 72

PAGE 73
