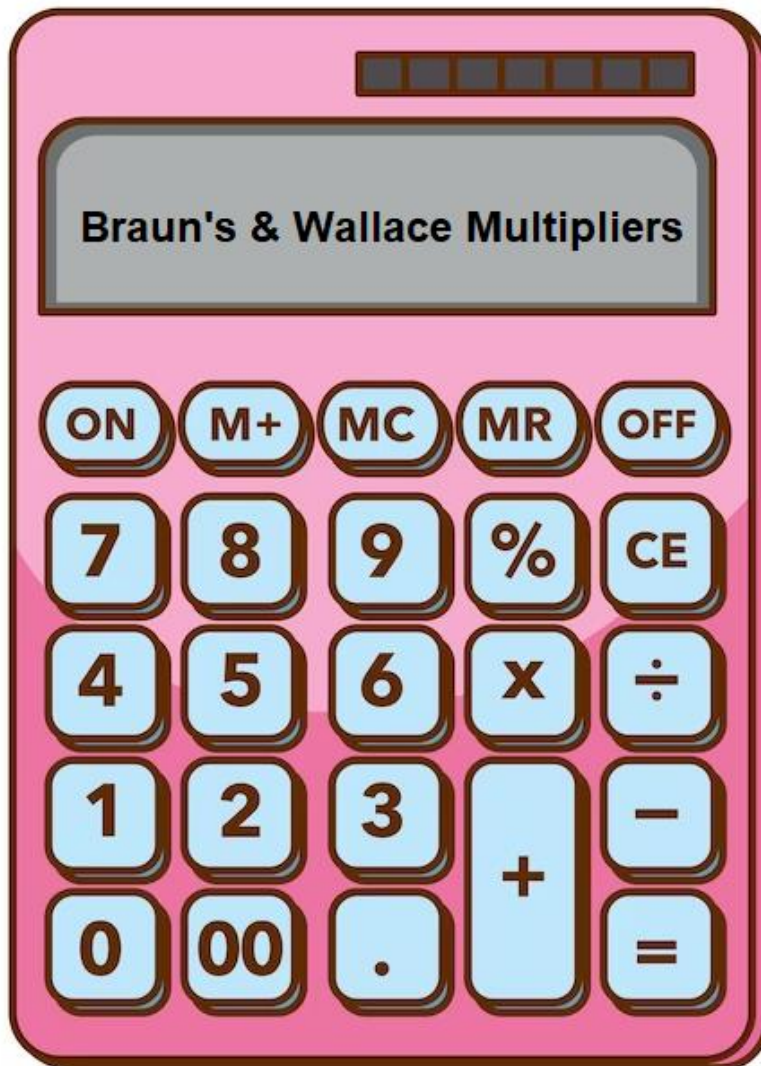


Electronics II

ACDA6001



Level 6 Assignment

Submitted By

Gareth Davies (2218705)

Contents

Table of Figures.....	3
Introduction	4
Project Research.....	5
Half Adder:	5
Full Adder:	5
Multipliers:	6
Brauns Multiplier:	6
Wallace Multiplier:.....	8
Multiplier Simulation	11
Brauns Simulation:.....	11
Brauns Timing:	12
Brauns Power Consumption:	15
Brauns Utilisation:	16
Brauns Summary:	16
Wallace Simulation:	19
Wallace Timing:	20
Wallace Power Consumption:	23
Wallace Utilisation:	24
Wallace Summary:	24
Multiplier Comparison.....	27
Conclusion	28
Bibliography	29
Appendix 1.0	30
Brauns Multiplier Program	30
Appendix 2.0	33
Wallace Multiplier Program	33

Table of Figures

Figure 1 - Half-Adder Structure and Truth Table [5]	5
Figure 2 - Full-Adder Structure and Truth Table [1]	5
Figure 3 - Configuration of an 8x8 Brauns Multiplier	7
Figure 4 Configuration of an 8x8 Wallace Multiplier	9
Figure 5 - Dot Diagram of an 8x8 Wallace Multiplier	10
Figure 6 – Hexadecimal Multiplication and Timing Diagram of A and C	11
Figure 7 – Hexadecimal Multiplication and Timing Diagram of 8 and 9	11
Figure 8 - 8x8 Brauns Multiplier Circuit Schematic.....	12
Figure 9 - Initial Timing Summary of Brauns Multiplier	12
Figure 10 - Brauns Multiplier Preliminary Combinational Delay	13
Figure 11 - Brauns Multiplier Delay Implementation	14
Figure 12 - Timing Summary with Time Constraint.....	14
Figure 13 - Combinational Delays with Timing Constraints Implemented.....	15
Figure 14 - Brauns Multiplier Power Usage	15
Figure 15 - Brauns Utilisation	16
Figure 16 - Summary of the Brauns Multiplier	16
Figure 17 - Device Routing for Brauns Implementation.....	17
Figure 18 - Brauns Multiplier I/O Planner	17
Figure 19 - RTL Analysis of 8x8 Brauns Multiplier	18
Figure 20 – Hexadecimal Multiplication and Timing Diagram of A and C	19
Figure 21 - Hexadecimal Multiplication and Timing Diagram of 8 and 9.....	19
Figure 22 - 8x8 Wallace Multiplier Circuit Schematic	20
Figure 23 - Initial Timing Summary of Wallace Multiplier	20
Figure 24 – Wallace Multiplier Preliminary Combinational Delay	21
Figure 25 - Wallace Multiplier Delay Implementation	22
Figure 26 - Timing Summary with Time Constraint.....	22
Figure 27 - Combinational Delays with Timing Constraints Implemented.....	23
Figure 28 - Wallace Multiplier Power Usage.....	23
Figure 29 - Wallace Utilisation	24
Figure 30 - Summary of the Wallace Multiplier	24
Figure 31 - Device Routing for Wallace Implementation	25
Figure 32 - Wallace Multiplier I/O Planner	25
Figure 33 - RTL Analysis of 8x8 Wallace Multiplier.....	26

Introduction

Binary multipliers are fundamental components in digital electronics that play a crucial role in performing many computing tasks. In the world of digital systems, binary multipliers are used. This report aims to explain and test two of the most common binary multipliers, the Brauns and Wallace.

This project has asked that 8x8 versions for each of these multipliers be designed and simulated to analysed performance characteristics. This will be done utilising Vivaldo software.

Project Research

Before discussing Brauns and Wallace multipliers, it is important to understanding their basic building blocks. These are half and full adders.

Half Adder:

A half adder is a digital circuit used to add two binary digits. It consists of two inputs, A and B, and will produce two outputs known as the sum and carry. To produce the sum, inputs A and B are fed into an XOR gate that performs addition. For the carry, inputs A and B are fed through an AND gate that performs multiplications. The sum and carry will then be combined similarly to algebraic addition to produce a final product.

The basic circuit diagram and truth table for a half adder can be seen below in Figure 1.

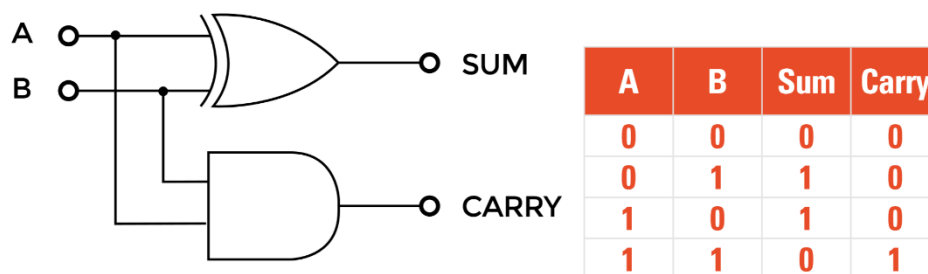


Figure 1 - Half-Adder Structure and Truth Table [5]

Full Adder:

A full adder consists of three inputs, A, B and Cin, and two outputs, the sum and carry. The sum is again produced via addition through a series of XOR gates. The carry is now produced via two AND gates and an OR gate before a final product is produced.

The structure and truth table for a full adder can be seen in Figure 2.

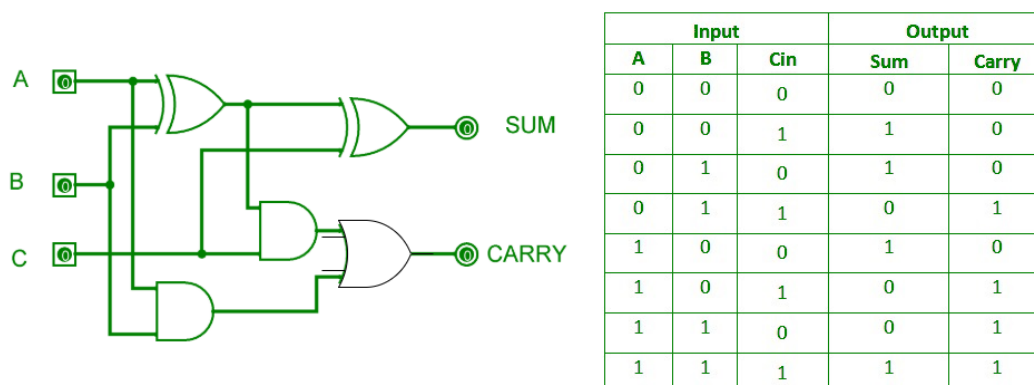


Figure 2 - Full-Adder Structure and Truth Table [1]

Multipliers:

Fundamentally, multipliers are used to find the product of two or more values. In computing, they are used to find the product of binary numbers. By using specific configurations of logic gates, half and full adders, binary multipliers can be created.

Two of the most efficient multipliers for this application are Brauns and Wallace multipliers.

Brauns Multiplier:

The Brauns multiplier, also known as the carry-save-array multiplier [2], is a simple parallel device that is limited to finding the product of two unsigned numbers. It is made up of an array of AND gates and adders arranged in a configuration that does not require logic registers for operation. Because of this, product bits can be generated simultaneously.

Each partial product can be combined with the previous sum of partial products, using a row of adders. This is done through a carry-save-adder that shifts carries one bit to the left [3].

The Least Significant Bit (LSB) is always fed into a half adder at the start of each row. The sum from the first half adder will represent a bit value within the final calculated product.

A carry-save-adder then feeds the carry into an adjacent adder to the left. The final carry of each row is fed into the input of the final adder of the proceeding row. The sum from each full adder is used as an input for the adder directly below in this array configuration.

This sequential addition continues until the final row of the multiplier. The sum from each adder within this row will represent a bit value within the final binary product. The final carry for this row will represent the Most Significant Bit (MSB) of the product.

The performance of the Brauns multiplier is excellent when working with unsigned operands that are less than 16-bit. It is fast, consumes little power and occupies a very small area due to its simplicity. This straightforward structure is also beneficial for expansion and fault finding within a 16-bit range.

The problem with Brauns multipliers is that their component requirement increases exponentially with the bit rate of the multiplier. Brauns multipliers for 16-bit plus operations are typically slow, inefficient and consume higher power compared to other multiplier configurations [2].

A flowchart highlighting the configuration of an 8x8 Brauns multiplier can be seen in Figure 3.

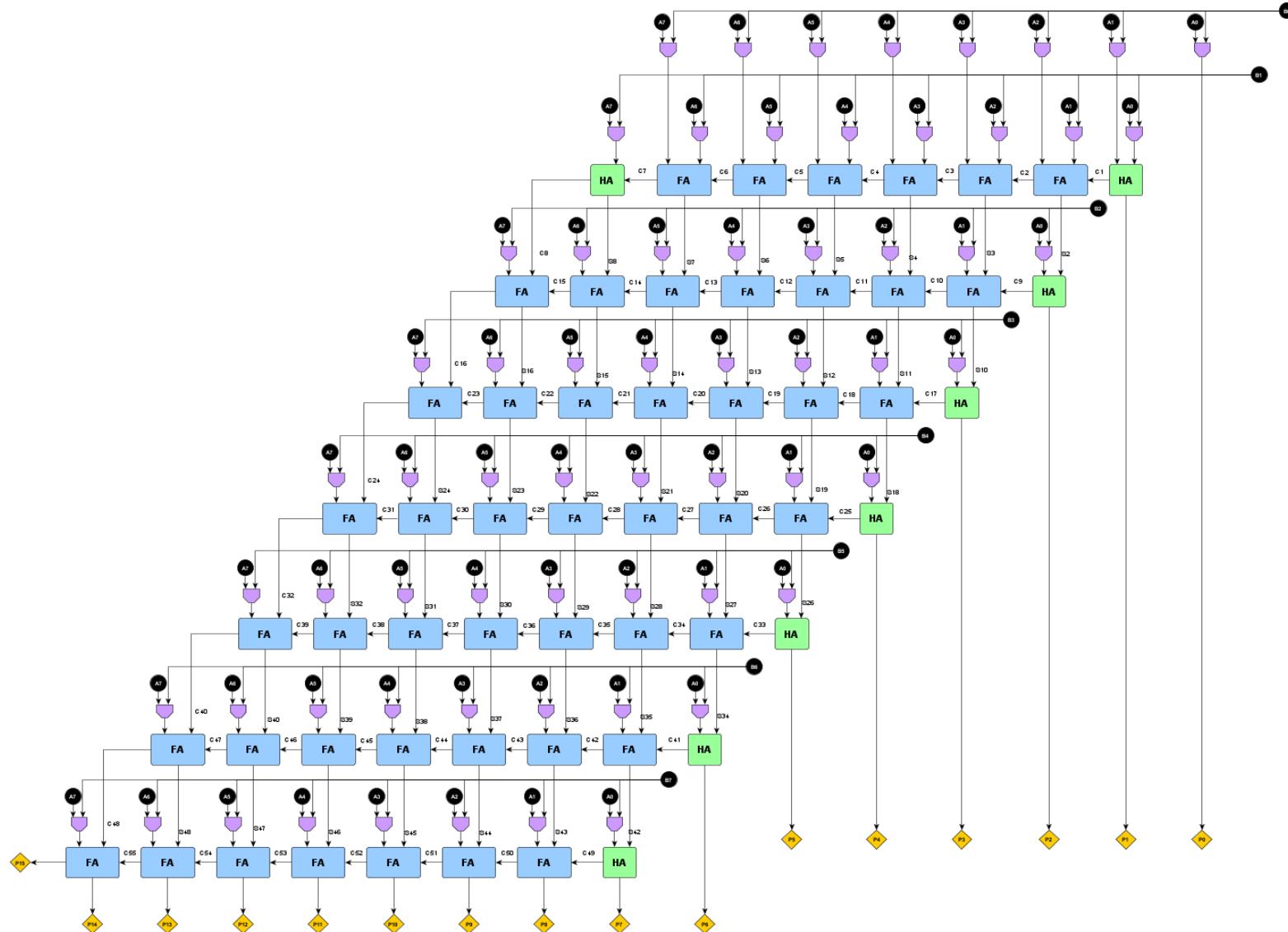


Figure 3 - Configuration of an 8x8 Brauns Multiplier

Wallace Multiplier:

The Wallace multiplier is another parallel multiplier that follows a similar array structure to the Brauns. There are however major differences between designs.

The Wallace multiplier generates fewer partial products compared to the Brauns multiplier. It achieves this by grouping operands into smaller blocks and generating the partial products of each block. This is known as partial product reduction [4].

Due to this, the carries are directly fed into the next stage of adders instead of laterally as seen in a Brauns configuration.

Despite circuit complexity increasing due to additional hardware requirements of partial product reduction, the overall efficiency of the multiplier improves as fewer partial products are calculated [4]. A Wallace multiplier is also scalable and doesn't suffer the same exponential setbacks through expansion as the Brauns device. This makes Wallace multiplier better suited to multiplication of larger numbers [2].

The additional hardware does however increase circuit size and complexity as well as overall power consumption of the device.

An example of an 8x8 Wallace multiplier can be seen in Figure 4. A dot diagram has also been created shown in Figure 5.

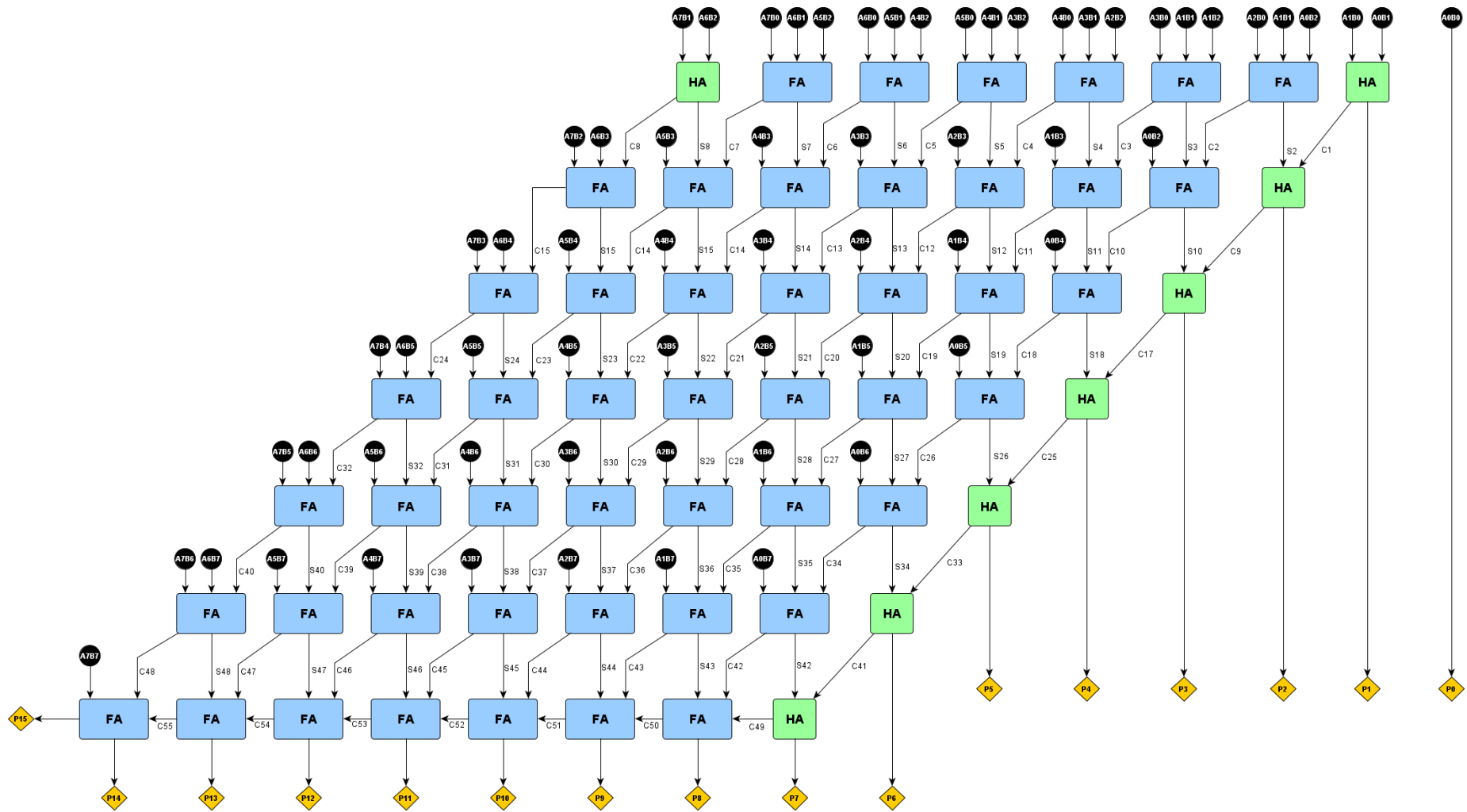


Figure 4 Configuration of an 8x8 Wallace Multiplier

8x8 Wallace Multiplier Dot Diagram

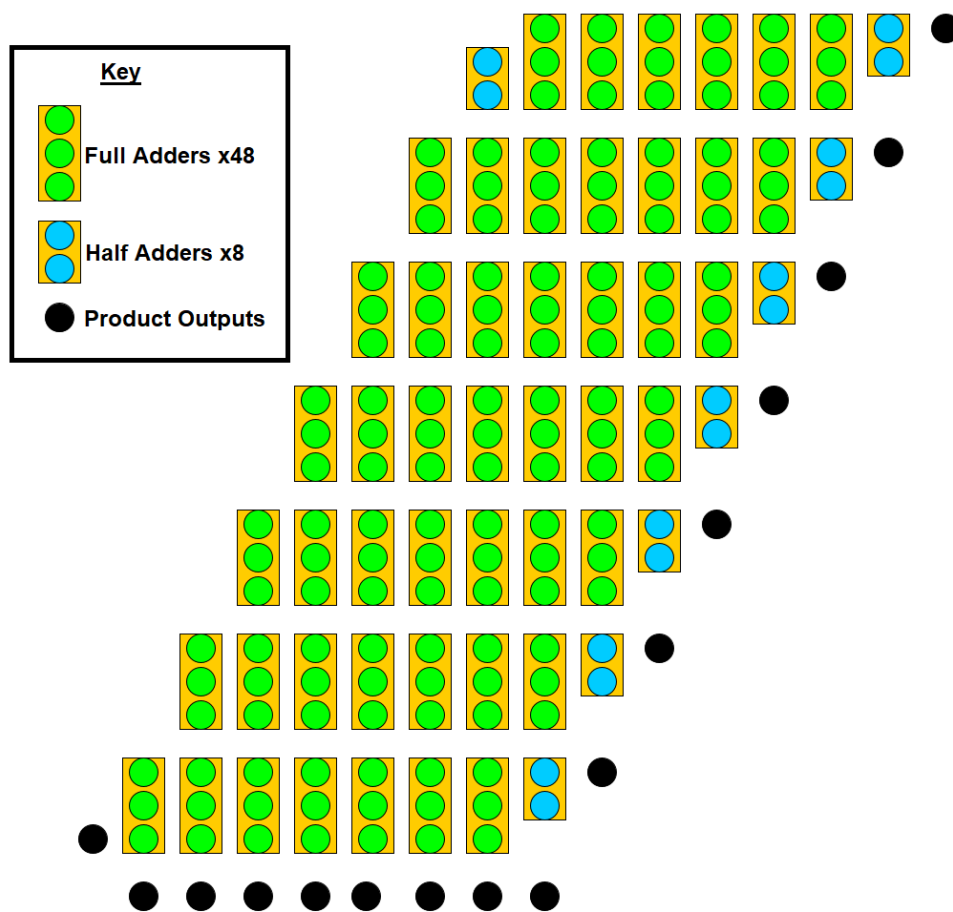


Figure 5 - Dot Diagram of an 8x8 Wallace Multiplier

Multiplier Simulation

With a fundamental understating of both Brauns and Wallace type multipliers, this project now asks that an 8x8 version for each is designed and tested utilising Vivaldo software.

The full programs created for each of these multipliers can be seen in Appendix 1.0 and 2.0.

Brauns Simulation:

Once the Brauns program was written, a simulation was conducted to confirm correct operation. Basic hexadecimal multiplication and timing diagram for this can be seen in Figures 5 and 6.

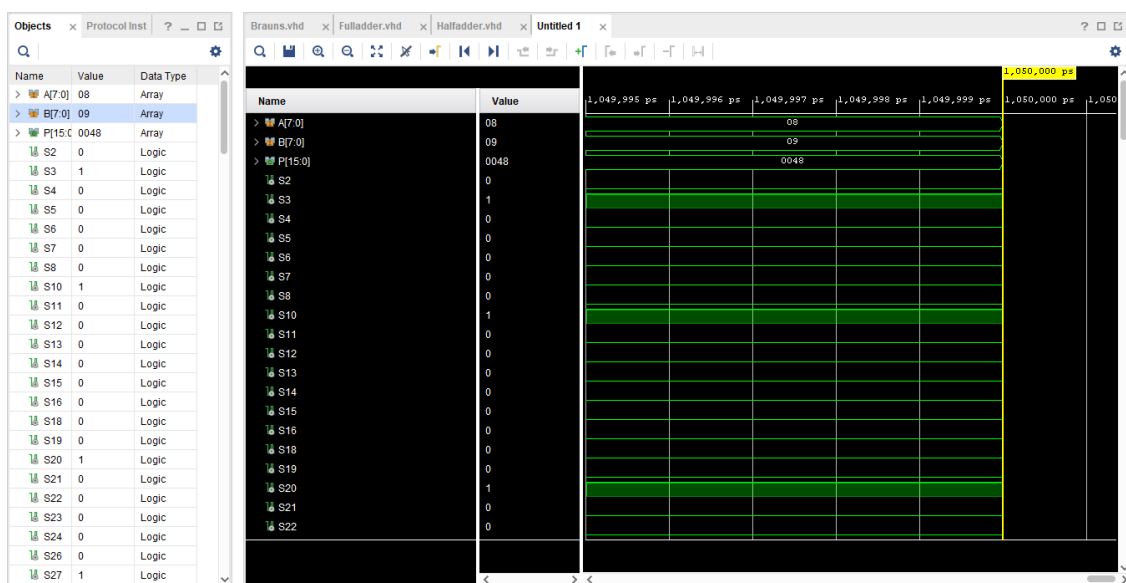


Figure 7 – Hexadecimal Multiplication and Timing Diagram of 8 and 9

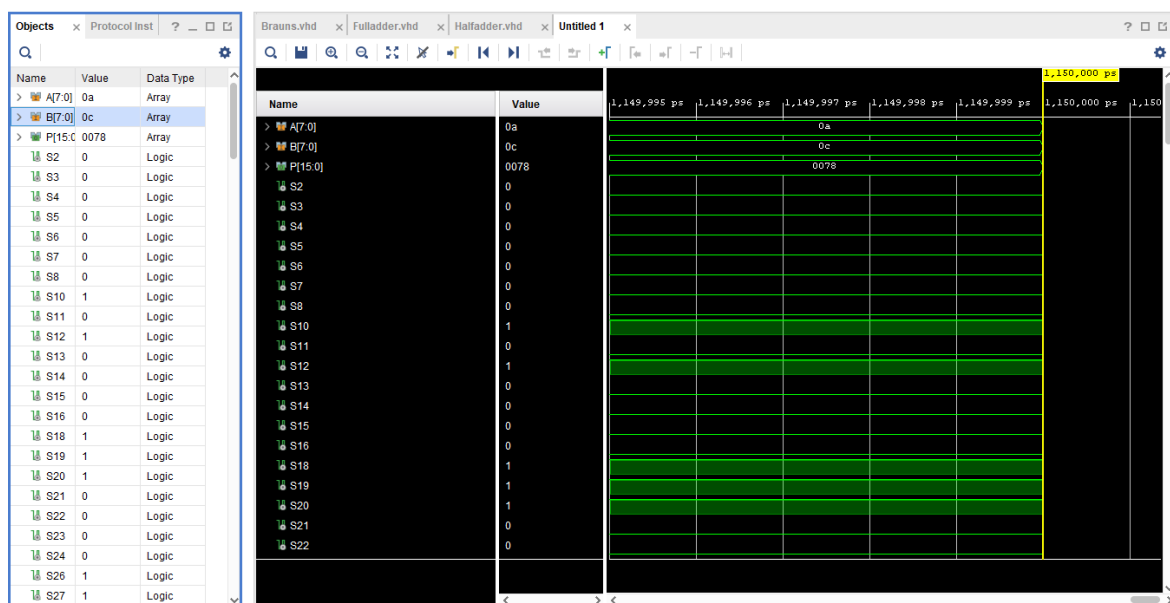


Figure 6 – Hexadecimal Multiplication and Timing Diagram of A and C

With correct operation confirmed, design synthesis was conducted and a simulated schematic of the Brauns multiplier was created as seen in Figure 8.

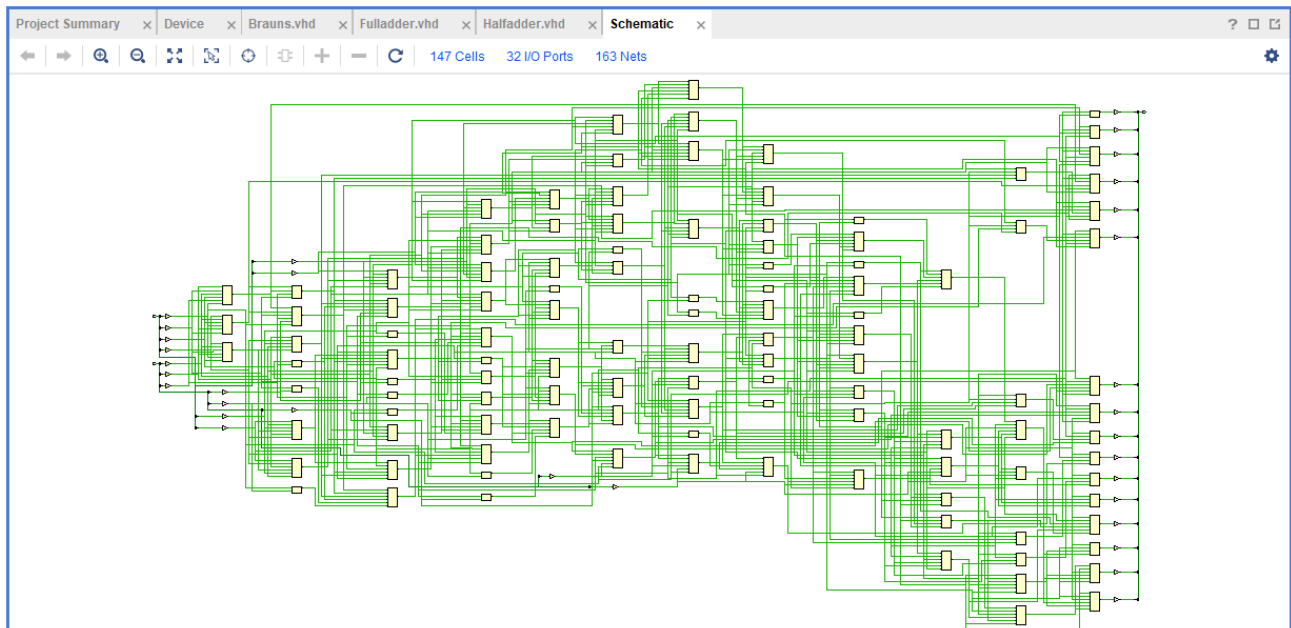


Figure 8 - 8x8 Brauns Multiplier Circuit Schematic

Brauns Timing:

The inputs and outputs (I/O) for this multiplier are now designated to specific pins to create nets within the multiplier. This allows for system implementation to be conducted to highlight timing data within the multiplier. The timing summary for this device can be seen in Figure 9 below.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	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

There are no user specified timing constraints.

Figure 9 - Initial Timing Summary of Brauns Multiplier

It can be seen that no time constraints are currently applied to the multiplier. This may cause operational delays and prevent the multiple from working efficiently.

Using the timing datasheet, the maximum delay within this device can be deduced. Time constrains can then be implemented to reduce this delay. The combinational delays for his simulation can be seen in Figure 10.

Tcl Console	Messages	Log	Reports	Design Runs	Timing x	Power	DRC	Package Pins	I/O Ports																																																																																				
Q General Information Timer Settings Design Timing Summary Methodology Summary > Check Timing (0) Intra-Clock Paths Inter-Clock Paths Other Path Groups User Ignored Paths > Unconstrained Paths ▼ Datasheet Input Ports Setup/Hold Output Ports Clock-to-out Combinational Delays Setup between Clocks					Q Combinational Delays <table><thead><tr><th>From Port</th><th>To Port</th><th>M a 1</th><th>Max Process Corner</th><th>Min Delay</th><th>Min Process Corner</th></tr></thead><tbody><tr><td>B[1]</td><td>P[13]</td><td>23.128</td><td>SLOW</td><td>6.496</td><td>FAST</td></tr><tr><td>A[2]</td><td>P[13]</td><td>23.076</td><td>SLOW</td><td>5.347</td><td>FAST</td></tr><tr><td>A[3]</td><td>P[13]</td><td>23.045</td><td>SLOW</td><td>5.307</td><td>FAST</td></tr><tr><td>B[0]</td><td>P[13]</td><td>22.925</td><td>SLOW</td><td>6.267</td><td>FAST</td></tr><tr><td>A[1]</td><td>P[13]</td><td>22.782</td><td>SLOW</td><td>5.600</td><td>FAST</td></tr><tr><td>B[2]</td><td>P[13]</td><td>22.713</td><td>SLOW</td><td>5.843</td><td>FAST</td></tr><tr><td>B[1]</td><td>P[14]</td><td>22.691</td><td>SLOW</td><td>6.247</td><td>FAST</td></tr><tr><td>A[2]</td><td>P[14]</td><td>22.639</td><td>SLOW</td><td>5.331</td><td>FAST</td></tr><tr><td>A[3]</td><td>P[14]</td><td>22.609</td><td>SLOW</td><td>5.291</td><td>FAST</td></tr><tr><td>A[5]</td><td>P[13]</td><td>22.574</td><td>SLOW</td><td>5.119</td><td>FAST</td></tr><tr><td>A[4]</td><td>P[13]</td><td>22.509</td><td>SLOW</td><td>5.290</td><td>FAST</td></tr><tr><td>B[0]</td><td>P[14]</td><td>22.489</td><td>SLOW</td><td>6.018</td><td>FAST</td></tr><tr><td>A[1]</td><td>P[14]</td><td>22.345</td><td>SLOW</td><td>5.584</td><td>FAST</td></tr></tbody></table>					From Port	To Port	M a 1	Max Process Corner	Min Delay	Min Process Corner	B[1]	P[13]	23.128	SLOW	6.496	FAST	A[2]	P[13]	23.076	SLOW	5.347	FAST	A[3]	P[13]	23.045	SLOW	5.307	FAST	B[0]	P[13]	22.925	SLOW	6.267	FAST	A[1]	P[13]	22.782	SLOW	5.600	FAST	B[2]	P[13]	22.713	SLOW	5.843	FAST	B[1]	P[14]	22.691	SLOW	6.247	FAST	A[2]	P[14]	22.639	SLOW	5.331	FAST	A[3]	P[14]	22.609	SLOW	5.291	FAST	A[5]	P[13]	22.574	SLOW	5.119	FAST	A[4]	P[13]	22.509	SLOW	5.290	FAST	B[0]	P[14]	22.489	SLOW	6.018	FAST	A[1]	P[14]	22.345	SLOW	5.584	FAST
From Port	To Port	M a 1	Max Process Corner	Min Delay	Min Process Corner																																																																																								
B[1]	P[13]	23.128	SLOW	6.496	FAST																																																																																								
A[2]	P[13]	23.076	SLOW	5.347	FAST																																																																																								
A[3]	P[13]	23.045	SLOW	5.307	FAST																																																																																								
B[0]	P[13]	22.925	SLOW	6.267	FAST																																																																																								
A[1]	P[13]	22.782	SLOW	5.600	FAST																																																																																								
B[2]	P[13]	22.713	SLOW	5.843	FAST																																																																																								
B[1]	P[14]	22.691	SLOW	6.247	FAST																																																																																								
A[2]	P[14]	22.639	SLOW	5.331	FAST																																																																																								
A[3]	P[14]	22.609	SLOW	5.291	FAST																																																																																								
A[5]	P[13]	22.574	SLOW	5.119	FAST																																																																																								
A[4]	P[13]	22.509	SLOW	5.290	FAST																																																																																								
B[0]	P[14]	22.489	SLOW	6.018	FAST																																																																																								
A[1]	P[14]	22.345	SLOW	5.584	FAST																																																																																								

Figure 10 - Brauns Multiplier Preliminary Combinational Delay

For this multiplier, it can be seen that a maximum delay of 23.128ns is present between input B1 and output P13. This maximum delay is also known as the critical path of the multiplier.

This critical path can also be approximated manually using the following assumptions:

1. AND gate time delay is equal to 1ns.
2. Each adder time delay is equal to 2ns.

Referring back to Figure 3, it can be seen that route B1 to P13 travels through seven AND gated and seven total adders. Hence the manually calculated critical path is:

$$Critical Path = (7 \times 1) + (7 \times 2) = 21ns$$

To eliminate the delay from this critical path, a time delay can be added between pins B1 and P13. The set-up for this timing delay can be seen in Figure 11.

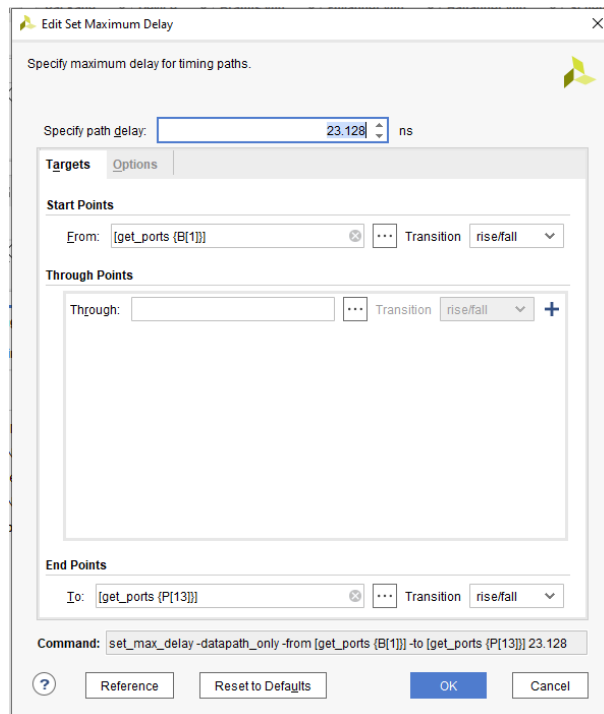


Figure 11 - Brauns Multiplier Delay Implementation

With this timing constrain, the device was re-implemented, and the following timing summary was created.

Tcl Console					Messages	Log	Reports	Design Runs	Timing	x	Power	DRC	Package Pins	I/O Ports
     					Design Timing Summary									
General Information														
Timer Settings														
Design Timing Summary														
Clock Summary (1)														
Methodology Summary														
Check Timing (1)														
Intra-Clock Paths														
Inter-Clock Paths														
Setup					Hold					Pulse Width				
Worst Negative Slack (WNS): 0.000 ns					Worst Hold Slack (WHS): inf					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: 1					Total Number of Endpoints: 16					Total Number of Endpoints: NA				
All user specified timing constraints are met.														

Figure 12 - Timing Summary with Time Constraint

This shows that there is now 0ns Worst Negative Slack (WNS) occurring as a result of the time constraints added. This can be further highlighted when inspecting the new combinational delays. A minimum delay of 0ns is now measure between B1 to P13 as shown in Figure 13.

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
B[1]	P[13]	23.128	SLOW	0.000	
B[1]	P[14]	22.691	SLOW	6.247	FAST
A[2]	P[14]	22.639	SLOW	5.331	FAST
A[3]	P[14]	22.609	SLOW	5.291	FAST
B[0]	P[14]	22.489	SLOW	6.018	FAST
A[1]	P[14]	22.345	SLOW	5.584	FAST
B[2]	P[14]	22.277	SLOW	5.594	FAST
A[5]	P[14]	22.137	SLOW	5.103	FAST
A[4]	P[14]	22.072	SLOW	5.274	FAST
B[1]	P[10]	21.974	SLOW	6.635	FAST
A[2]	P[10]	21.929	SLOW	4.997	FAST
B[1]	P[12]	21.915	SLOW	6.121	FAST
A[3]	P[10]	21.899	SLOW	4.670	FAST
A[2]	P[12]	21.863	SLOW	5.202	FAST

Figure 13 - Combinational Delays with Timing Constraints Implemented

Brauns Power Consumption:

In addition to a timing summary, power consumption from this multiplier can also be analysed.

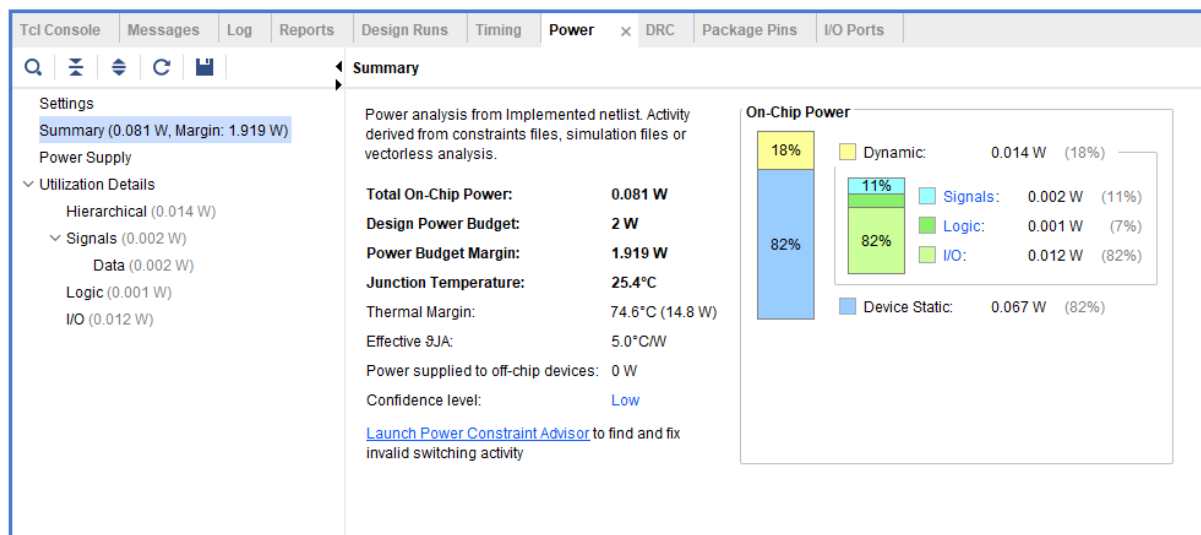


Figure 14 - Brauns Multiplier Power Usage

Of the specified 2W power budget, Figure 14 shows that only 0.081W is being utilised by the Brauns multiplier.

Brauns Utilisation:

With the final implementation complete, the utilisation tool can now be used to highlight device usage of its Look-Up Tables (LUTs) and I/O components. From Figure 15, it can be seen that this Brauns multiplier utilises 90 LUTs and 27 I/O ports.

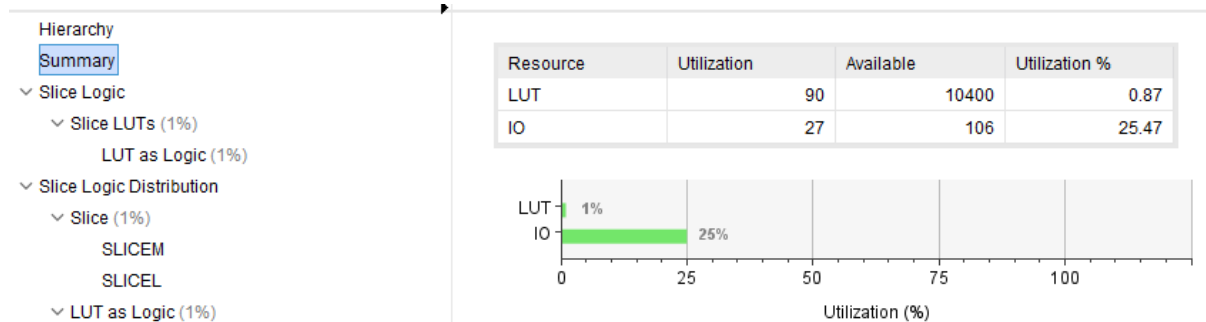


Figure 15 - Brauns Utilisation

Brauns Summary:

The figures within this section summaries the final design of this 8x8 Brauns multiplier.

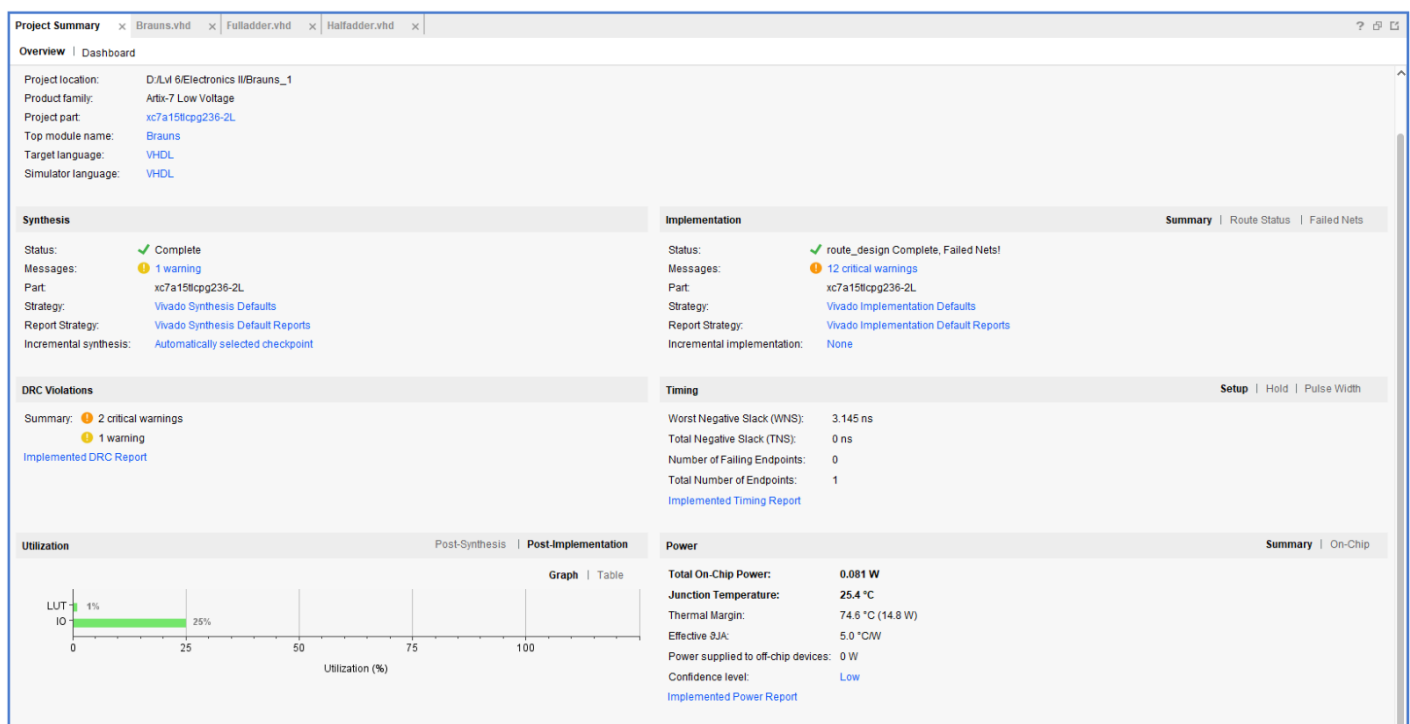


Figure 16 - Summary of the Brauns Multiplier

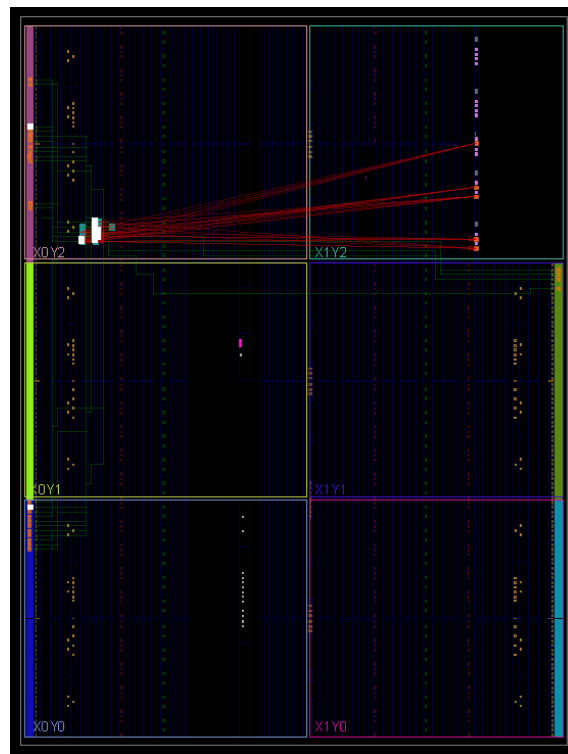


Figure 17 - Device Routing for Brauns Implementation

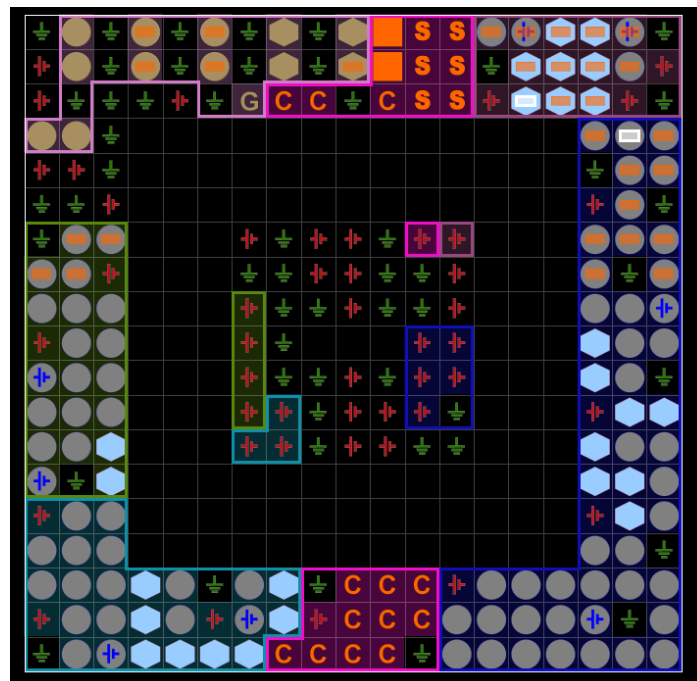


Figure 18 - Brauns Multiplier I/O Planner

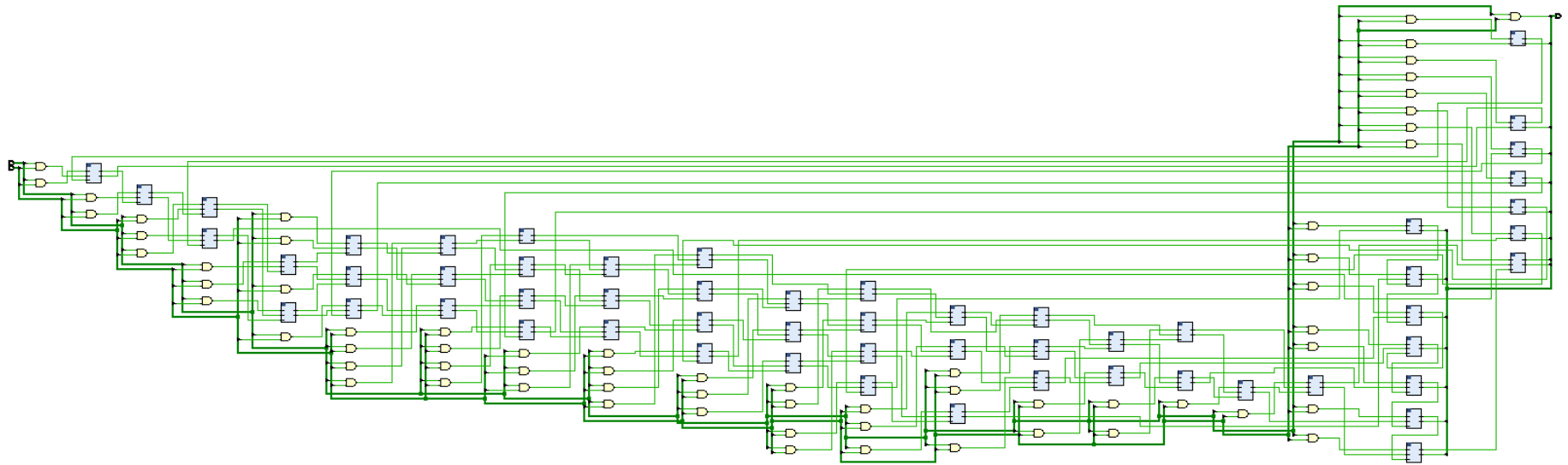


Figure 19 - RTL Analysis of 8x8 Brauns Multiplier

Wallace Simulation:

Using an identical method of that described for the Brauns, the designed Wallace 8x8 program was then simulated, synthesised, and implemented.

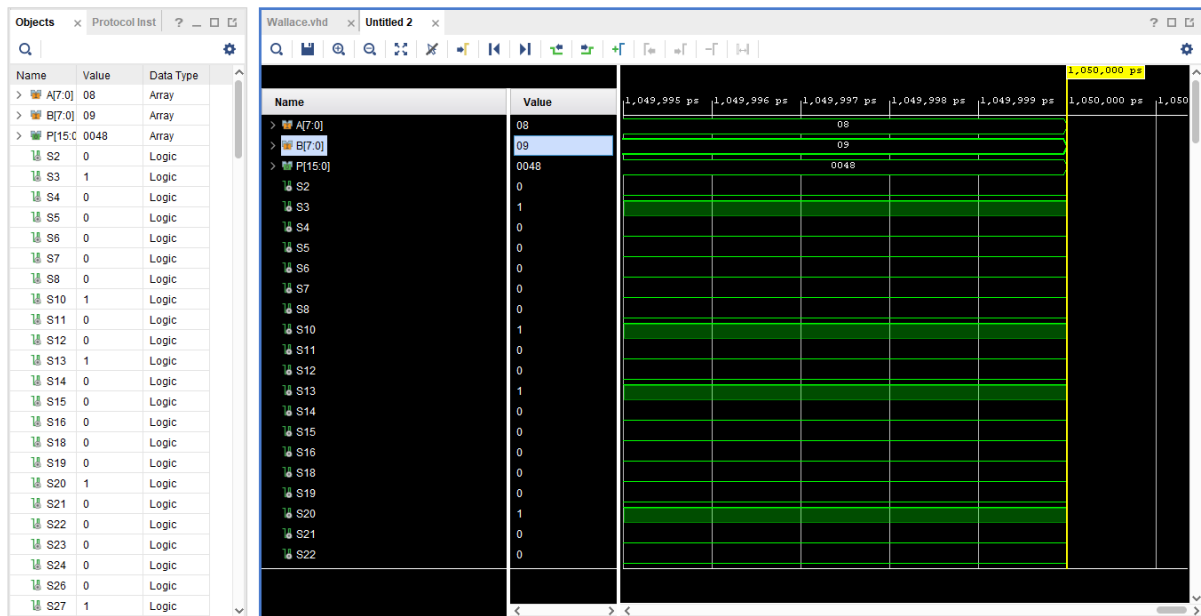


Figure 21 - Hexadecimal Multiplication and Timing Diagram of 8 and 9

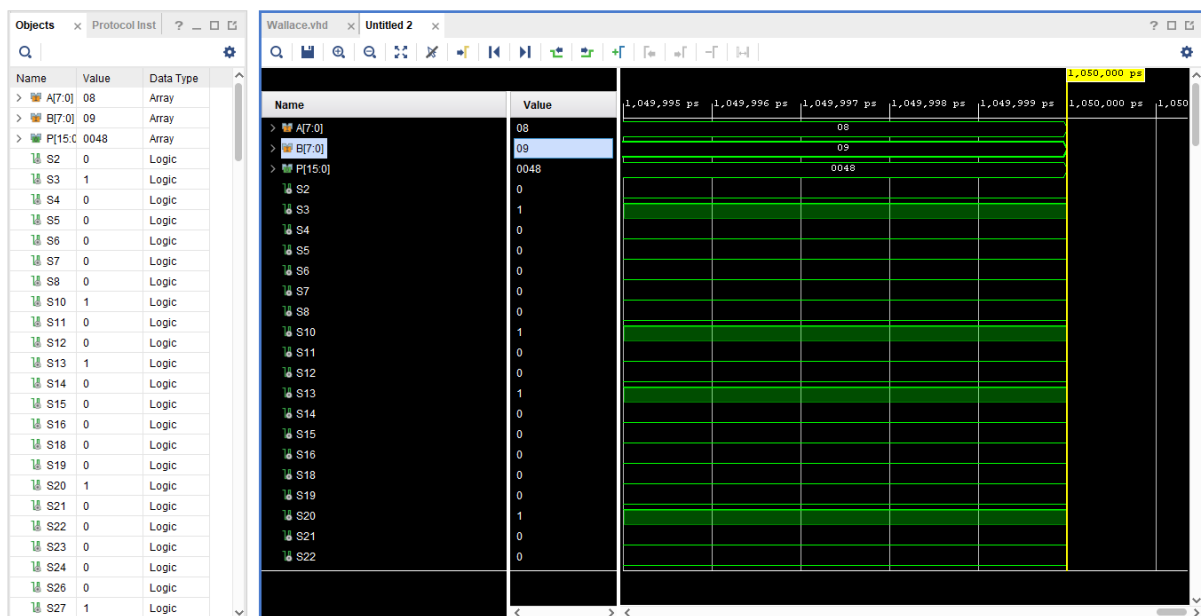


Figure 20 – Hexadecimal Multiplication and Timing Diagram of A and C

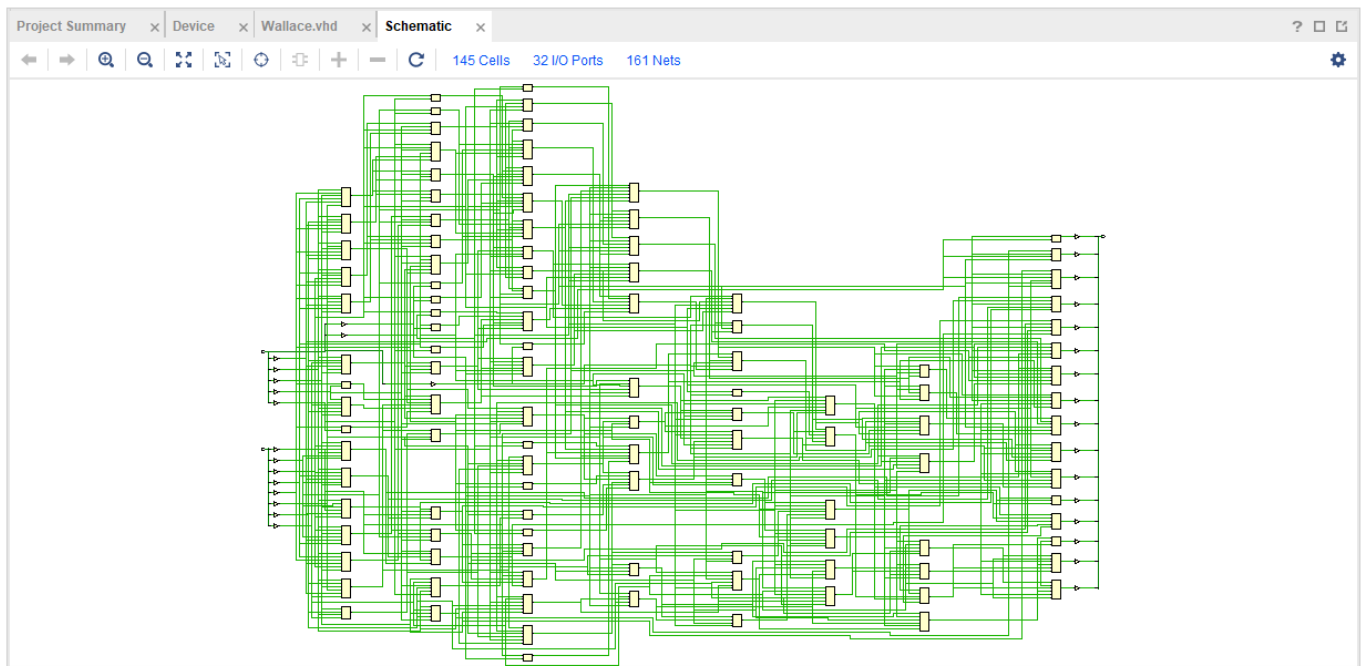


Figure 22 - 8x8 Wallace Multiplier Circuit Schematic

Wallace Timing:

Design Timing Summary			
Setup	Hold	Pulse Width	
Worst Negative Slack (WNS):	Inf	Worst Hold Slack (WHS):	Inf
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	16	Total Number of Endpoints:	16
There are no user specified timing constraints.			

Figure 23 - Initial Timing Summary of Wallace Multiplier

Tcl Console

Messages

Log

Reports

Design Runs

Timing

Power

DRC

Package Pins

I/O Ports

Q

⌵

⌶

⌷

⌸

⌹

⌺

Q

Combinational Delays

General Information

Timer Settings

Design Timing Summary

Methodology Summary

> Check Timing (0)

Intra-Clock Paths

Inter-Clock Paths

Other Path Groups

User Ignored Paths

> Unconstrained Paths

▼ Datasheet

Input Ports Setup/Hold

Output Ports Clock-to-out

Combinational Delays

Setup between Clocks

From Port

To Port

Ma

1

Max Process Corner

Min Delay

Min Process Corner

A[4]

P[10]

17.987

SLOW

5.735

FAST

B[0]

P[10]

17.690

SLOW

5.412

FAST

A[5]

P[10]

17.658

SLOW

5.750

FAST

A[4]

P[13]

17.567

SLOW

5.067

FAST

B[0]

P[13]

17.552

SLOW

5.713

FAST

A[3]

P[10]

17.541

SLOW

4.909

FAST

A[4]

P[12]

17.301

SLOW

4.898

FAST

B[0]

P[12]

17.286

SLOW

5.548

FAST

A[5]

P[13]

17.238

SLOW

5.298

FAST

A[3]

P[13]

17.121

SLOW

5.002

FAST

A[2]

P[13]

17.056

SLOW

5.349

FAST

A[2]

P[10]

17.017

SLOW

4.506

FAST

B[1]

P[10]

16.992

SLOW

5.158

FAST

A[5]

P[12]

16.972

SLOW

5.271

FAST

A[4]

P[11]

16.880

SLOW

4.984

FAST

B[0]

P[11]

16.865

SLOW

5.404

FAST

A[3]

P[12]

16.855

SLOW

4.838

FAST

A[2]

P[12]

16.790

SLOW

5.185

FAST

B[2]

P[10]

16.764

SLOW

5.358

FAST

B[2]

P[13]

16.704

SLOW

5.255

FAST

Figure 24 – Wallace Multiplier Preliminary Combinational Delay

It can be seen that a maximum delay of 17.987ns is present between input A4 and output P10 of this Wallace multiplier. This is its critical path.

This critical path can also be approximated manually using the following assumptions:

1. AND gate time delay is equal to 1ns.
2. Each adder time delay is equal to 2ns.

Referring back to Figure 4, it can be seen that route A4 to P10 requires one AND gate and seven total adders. Hence the manually calculated critical path is:

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

To eliminate the delay from this critical path, a time delay can be added between pins A4 and P10. The set-up for this timing delay can be seen in Figure 25.

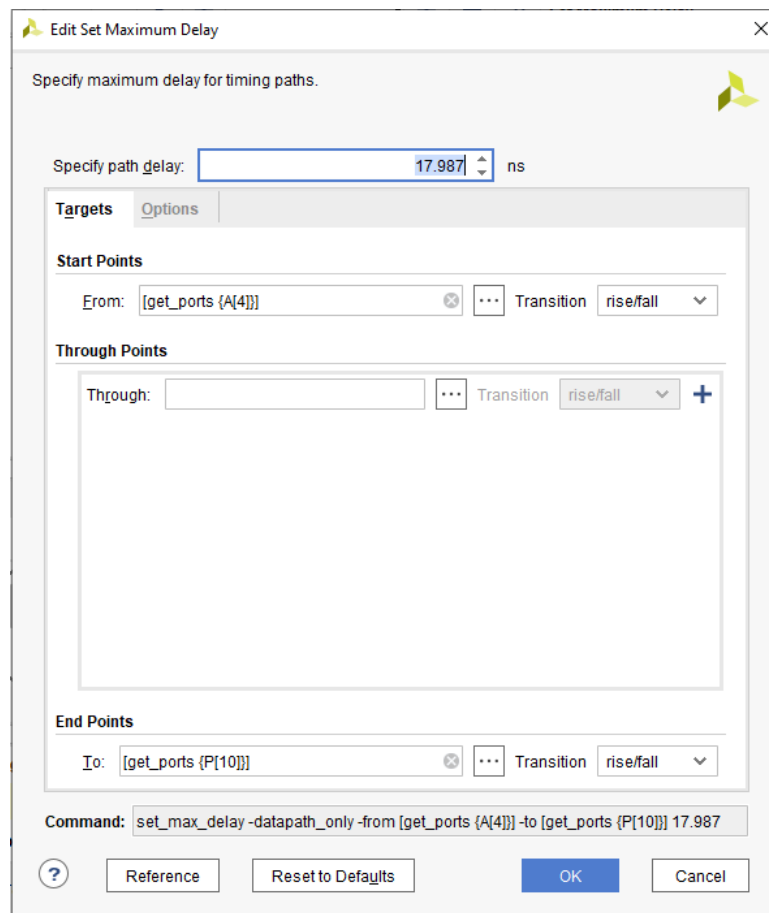


Figure 25 - Wallace Multiplier Delay Implementation

Design Timing Summary			
Setup	Hold	Pulse Width	
Worst Negative Slack (WNS): 0.000 ns	Worst Hold Slack (WHS): inf	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: 1	Total Number of Endpoints: 16	Total Number of Endpoints:	NA
All user specified timing constraints are met.			

Figure 26 - Timing Summary with Time Constraint

Figure 26 shows that the timing constraints added have reduced the WNS to 0ns. The combinational delays shown in Figure 27 also confirm a 0ns delay on the critical path.

Tcl Console	Messages	Log	Reports	Design Runs	Timing	Power	DRC	Package Pins	I/O Ports
Combinational Delays									
General Information	From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner			
Timer Settings	A[4]	P[10]	17.987	SLOW	0.000				
Design Timing Summary	A[4]	P[13]	17.567	SLOW	5.067	FAST			
Clock Summary (1)	B[0]	P[13]	17.552	SLOW	5.713	FAST			
Methodology Summary	A[4]	P[12]	17.301	SLOW	4.898	FAST			
> Check Timing (1)	B[0]	P[12]	17.286	SLOW	5.548	FAST			
Intra-Clock Paths	A[5]	P[13]	17.238	SLOW	5.298	FAST			
Inter-Clock Paths	A[3]	P[13]	17.121	SLOW	5.002	FAST			
> Other Path Groups	A[2]	P[13]	17.056	SLOW	5.349	FAST			
User Ignored Paths	A[5]	P[12]	16.972	SLOW	5.271	FAST			
Unconstrained Paths	A[4]	P[11]	16.880	SLOW	4.984	FAST			
Datasheet	B[0]	P[11]	16.865	SLOW	5.404	FAST			
Input Ports Setup/Hold	A[3]	P[12]	16.855	SLOW	4.838	FAST			
Output Ports Clock-to-out	A[2]	P[12]	16.790	SLOW	5.185	FAST			
Combinational Delays	B[2]	P[13]	16.704	SLOW	5.255	FAST			
Setup between Clocks	A[4]	P[15]	16.669	SLOW	4.416	FAST			
	A[4]	P[9]	16.661	SLOW	5.810	FAST			
	B[0]	P[15]	16.654	SLOW	5.062	FAST			

Figure 27 - Combinational Delays with Timing Constraints Implemented

Wallace Power Consumption:

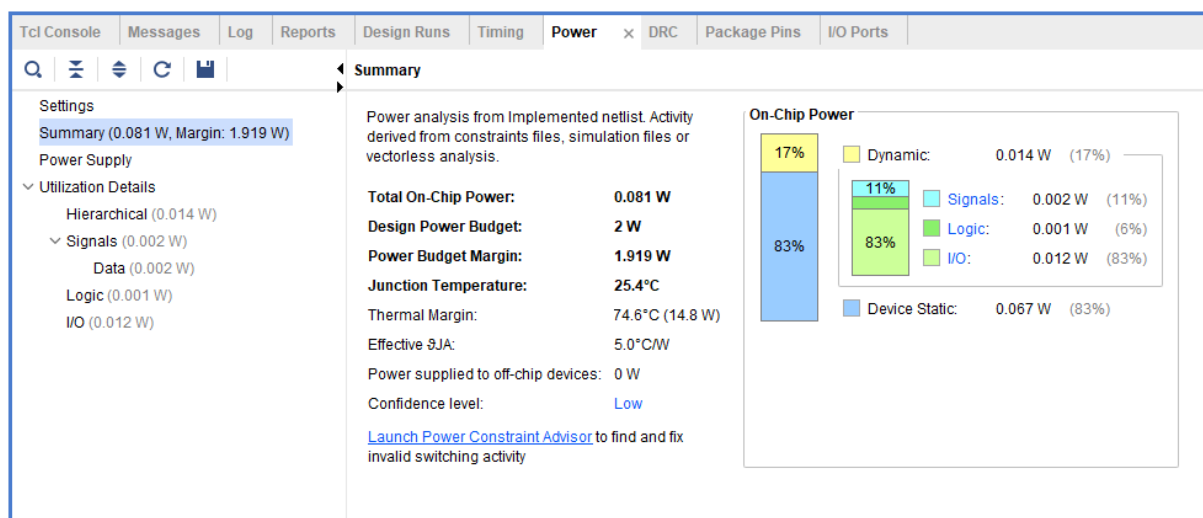


Figure 28 - Wallace Multiplier Power Usage

Despite using a different multiplication method, Figure 26 highlights a power consumption of 0.081W for this Wallace configuration, identical that of Brauns.

Wallace Utilisation:

The utilisation tool shows that the Wallace multiplier requires 86 LUTs and 26 I/O port to operate as shown in Figure 29.

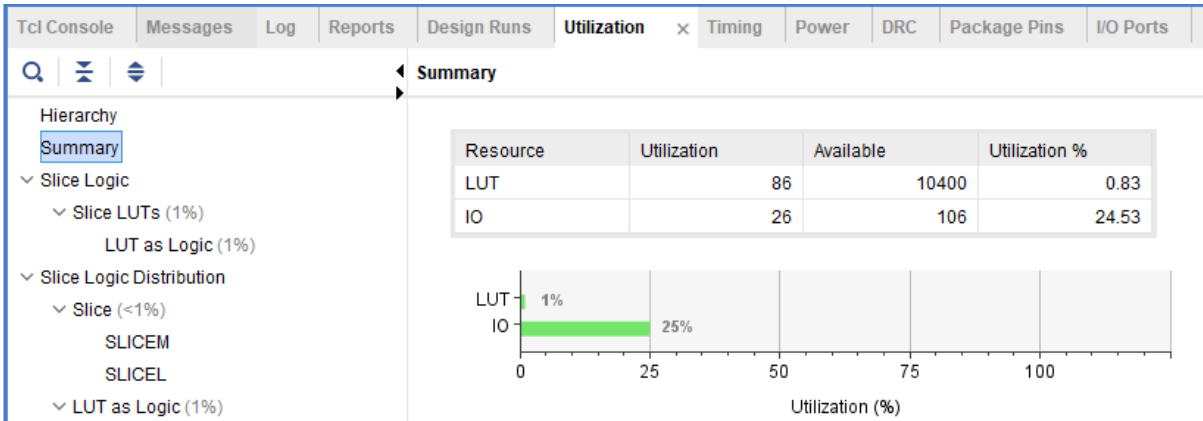


Figure 29 - Wallace Utilisation

Wallace Summary:

The figures within this section summaries the final design of this 8x8 Wallace multiplier.

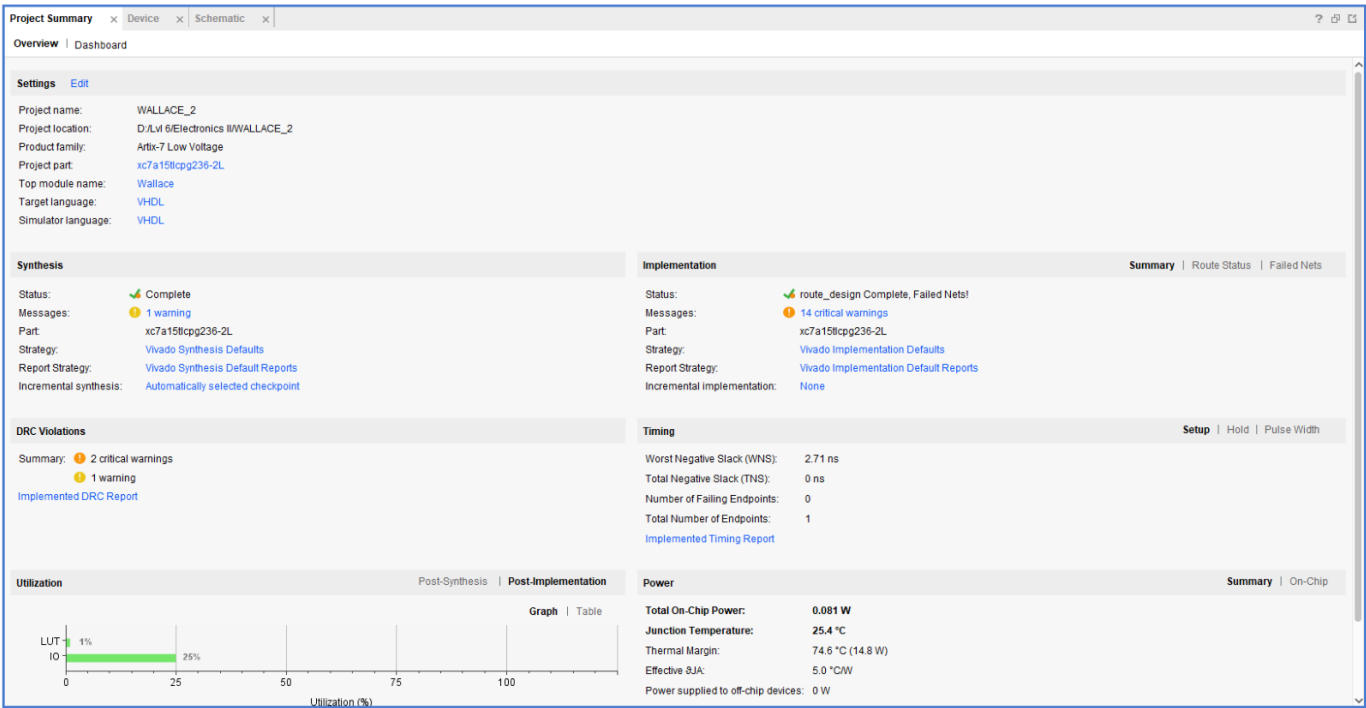


Figure 30 - Summary of the Wallace Multiplier

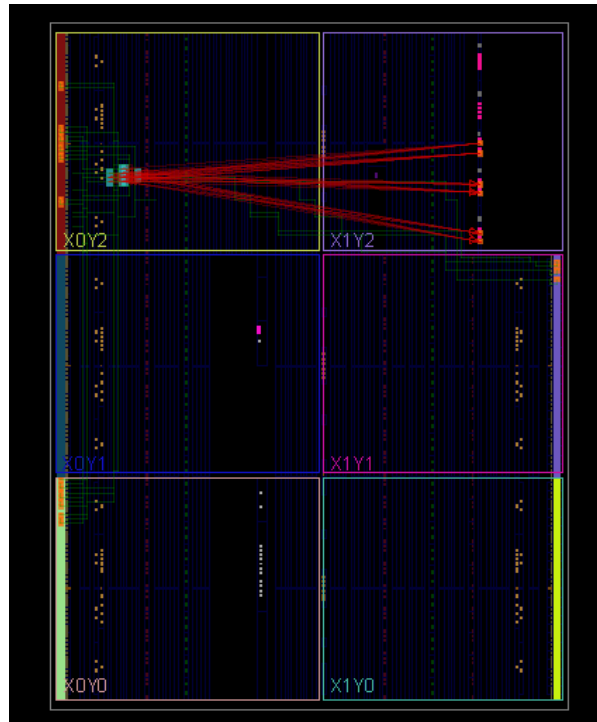


Figure 31 - Device Routing for Wallace Implementation

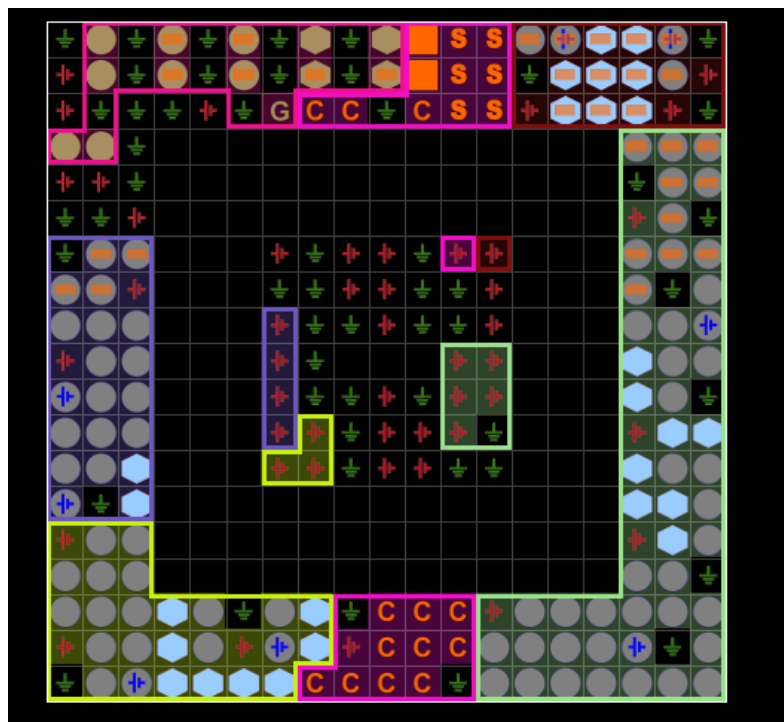


Figure 32 - Wallace Multiplier I/O Planner

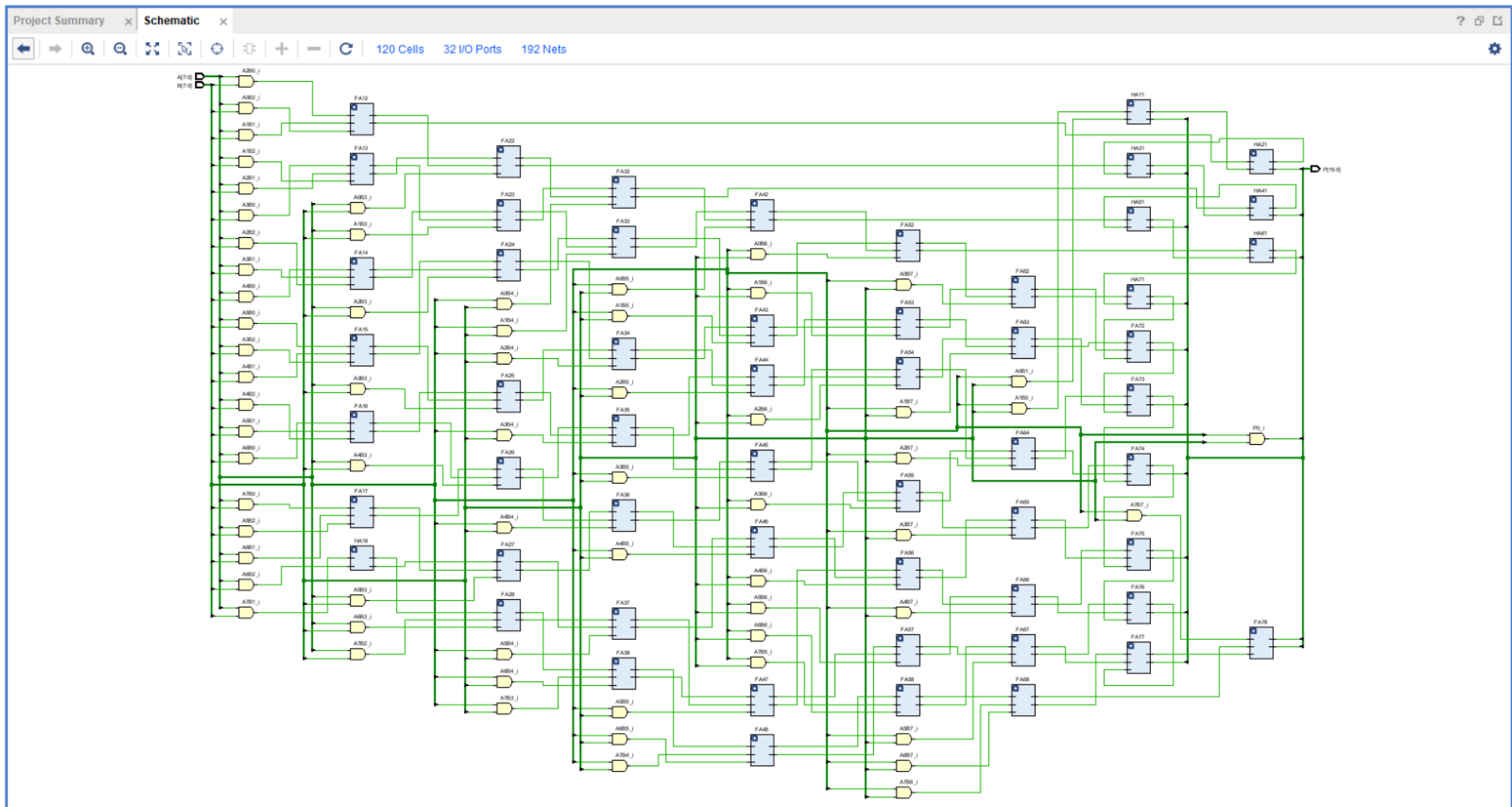


Figure 33 - RTL Analysis of 8x8 Wallace Multiplier

Multiplier Comparison

Table 1 is a comparison table used highlight the key differences between the Brauns and Wallace multiplier simulations.

Multiplier	Critical Path Delay	Power Consumption	LUTs	I/O
Brauns	23.128 ns	0.081 W	90	27
Wallace	17.987 ns	0.081 W	86	26

Table 1 - 8x8 Multiplier Comparison

From Table 1, it can be seen that the simulated multipliers follow the general trends outlined in the project research.

The critical path delay measured for the Brauns multiplier is nearly 33% longer compared to that experienced by the Wallace. This is due to the partial products being calculated for each bit pair within the Brauns. The partial product reduction of the Wallace equivalent reduces this delay.

Despite research suggesting the Wallace multiplier would consume additional power due to hardware requirements, it can be seen that power consumption is identical during simulation. It is thought that this has occurred due to the relatively small size of the 8-bit multipliers. Should a 16 or 32-bit multiplier be used, it is believed a discrepancy between powers would be seen.

The LUTs and I/O required for the Brauns multiplier is seen to be slightly higher than that of the Wallace. This is due to the AND gate requirements for partial product calculations of every bit pair within Brauns configuration. This difference would again get significantly wider if the multipliers size was increased.

Conclusion

The 8x8 multipliers designed and simulated for this project have been used to support project research highlighting the difference between Brauns and Wallace configurations.

The Braun multiplier has been found to be ideal for small, less than 16-bit applications, that required efficient area utilisation and low power consumption. Due to its additional gate requirements, this is achieved at the cost of operational speed.

On the other hand, Wallace multipliers are ideal for higher speed applications facilitated through the use of partial product reduction. They are also better suited to the multiplication of larger numbers due to their scalability. The downside to this configuration is that the circuit complexity increases device size and the overall power consumption of the device.

Whilst basic trends have been found through these simulations, the power consumption and device requirements of the 8x8 Brauns and Wallace multipliers are very similar. It is believed more decisive results would be achieved through the simulation of 16 and 32-bit versions of the multipliers.

Moving forward, expansion into these larger multipliers would be recommended. Additionally, both multiplying techniques can be configured in slightly varying methods for different applications. Research and simulation of these new methods may provide better insight into the overall operational characteristic for each type of multiplier.

Bibliography

- [1] Geeks for Geeks, "Difference between Half adder and full adder," [Online]. Available: <https://www.geeksforgeeks.org/difference-between-half-adder-and-full-adder/>. [Accessed 05 01 2024].
- [2] M. V. M. C. B.Chennaiah, "Performance Analysis of Different Multipliers," International Journal of Research , 2018.
- [3] E. F. Brickell, A Fast Modular Multiplication Algorithm With Application To Two Key Cryptography, Plenum, 1982.
- [4] K. G. S. Himanshu Bansal, "Wallace Tree Multiplier Designs: A Performance Comparison," University, Lakshmangarh, Sikar, Rajasthan, India, 2014.
- [5] RaspberryPie, "Building a Half Adder," [Online]. Available: <https://projects.raspberrypi.org/en/projects/halfadder>. [Accessed 05 01 2024].

Appendix 1.0

Brauns Multiplier Program

```
20 :
21 :
22 : library IEEE;
23 : use IEEE.STD_LOGIC_1164.ALL;
24 :
25 : -- Uncomment the following library declaration if using
26 : -- arithmetic functions with Signed or Unsigned values
27 : --use IEEE.NUMERIC_STD.ALL;
28 :
29 : -- Uncomment the following library declaration if instantiating
30 : -- any Xilinx leaf cells in this code.
31 : --library UNISIM;
32 : --use UNISIM.VComponents.all;
33 :
34 : entity Brauns is
35 :     Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
36 :           B : in STD_LOGIC_VECTOR (7 downto 0);
37 :           P : out STD_LOGIC_VECTOR (15 downto 0));
38 : end Brauns;
39 :
40 : architecture Behavioral of Brauns is
41 :
42 :
43 : component Fulladder is
44 :     Port ( A : in STD_Logic;
45 :           B : in STD_Logic;
46 :           Cin : in STD_Logic;
47 :           S : out STD_Logic;
48 :           Cout : out STD_Logic);
49 : end component;
50 :
51 : component Halfadder is
52 :     Port ( A : in STD_LOGIC;
53 :           B : in STD_LOGIC;
54 :           S : out STD_LOGIC;
55 :           Cout : out STD_LOGIC);
56 : end component;
57 :
58 : 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;
59 : 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,C26,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;
60 :
```

```

60
61 signal A1B0 : STD_Logic;
62 signal A2B0 : STD_Logic;
63 signal A3B0 : STD_Logic;
64 signal A4B0 : STD_Logic;
65 signal A5B0 : STD_Logic;
66 signal A6B0 : STD_Logic;
67 signal A7B0 : STD_Logic;
68
69 signal A0B1 : STD_Logic;
70 signal A1B1 : STD_Logic;
71 signal A2B1 : STD_Logic;
72 signal A3B1 : STD_Logic;
73 signal A4B1 : STD_Logic;
74 signal A5B1 : STD_Logic;
75 signal A6B1 : STD_Logic;
76 signal A7B1 : STD_Logic;
77
78 signal A0B2 : STD_Logic;
79 signal A1B2 : STD_Logic;
80 signal A2B2 : STD_Logic;
81 signal A3B2 : STD_Logic;
82 signal A4B2 : STD_Logic;
83 signal A5B2 : STD_Logic;
84 signal A6B2 : STD_Logic;
85 signal A7B2 : STD_Logic;
86
87 signal A0B3 : STD_Logic;
88 signal A1B3 : STD_Logic;
89 signal A2B3 : STD_Logic;
90 signal A3B3 : STD_Logic;
91 signal A4B3 : STD_Logic;
92 signal A5B3 : STD_Logic;
93 signal A6B3 : STD_Logic;
94 signal A7B3 : STD_Logic;
95
96 signal A0B4 : STD_Logic;
97 signal A1B4 : STD_Logic;
98 signal A2B4 : STD_Logic;
99 signal A3B4 : STD_Logic;
100 signal A4B4 : STD_Logic;
101 signal A5B4 : STD_Logic;
102 signal A6B4 : STD_Logic;
103 signal A7B4 : STD_Logic;
104
105 signal A0B5 : STD_Logic;
106 signal A1B5 : STD_Logic;
107 signal A2B5 : STD_Logic;
108 signal A3B5 : STD_Logic;
109 signal A4B5 : STD_Logic;
110 signal A5B5 : STD_Logic;
111 signal A6B5 : STD_Logic;
112 signal A7B5 : STD_Logic;
113
114 signal A0B6 : STD_Logic;
115 signal A1B6 : STD_Logic;
116 signal A2B6 : STD_Logic;
117 signal A3B6 : STD_Logic;
118 signal A4B6 : STD_Logic;
119 signal A5B6 : STD_Logic;
120 signal A6B6 : STD_Logic;
121 signal A7B6 : STD_Logic;
122
123 signal A0B7 : STD_Logic;
124 signal A1B7 : STD_Logic;
125 signal A2B7 : STD_Logic;
126 signal A3B7 : STD_Logic;
127 signal A4B7 : STD_Logic;
128 signal A5B7 : STD_Logic;
129 signal A6B7 : STD_Logic;
130 signal A7B7 : STD_Logic;
131

```

```

132 begin
133
134 A1B0 <= A(1) AND B(0);
135 A2B0 <= A(2) AND B(0);
136 A3B0 <= A(3) AND B(0);
137 A4B0 <= A(4) AND B(0);
138 A5B0 <= A(5) AND B(0);
139 A6B0 <= A(6) AND B(0);
140 A7B0 <= A(7) AND B(0);
141
142 A0B1 <= A(0) AND B(1);
143 A1B1 <= A(1) AND B(1);
144 A2B1 <= A(2) AND B(1);
145 A3B1 <= A(3) AND B(1);
146 A4B1 <= A(4) AND B(1);
147 A5B1 <= A(5) AND B(1);
148 A6B1 <= A(6) AND B(1);
149 A7B1 <= A(7) AND B(1);
150
151 A0B2 <= A(0) AND B(2);
152 A1B2 <= A(1) AND B(2);
153 A2B2 <= A(2) AND B(2);
154 A3B2 <= A(3) AND B(2);
155 A4B2 <= A(4) AND B(2);
156 A5B2 <= A(5) AND B(2);
157 A6B2 <= A(6) AND B(2);
158 A7B2 <= A(7) AND B(2);
159
160 A0B3 <= A(0) AND B(3);
161 A1B3 <= A(1) AND B(3);
162 A2B3 <= A(2) AND B(3);
163 A3B3 <= A(3) AND B(3);
164 A4B3 <= A(4) AND B(3);
165 A5B3 <= A(5) AND B(3);
166 A6B3 <= A(6) AND B(3);
167 A7B3 <= A(7) AND B(3);
168
169 A0B4 <= A(0) AND B(4);
170 A1B4 <= A(1) AND B(4);
171 A2B4 <= A(2) AND B(4);
172 A3B4 <= A(3) AND B(4);
173 A4B4 <= A(4) AND B(4);
174 A5B4 <= A(5) AND B(4);
175 A6B4 <= A(6) AND B(4);
176 A7B4 <= A(7) AND B(4);
177
178 A0B5 <= A(0) AND B(5);
179 A1B5 <= A(1) AND B(5);
180 A2B5 <= A(2) AND B(5);
181 A3B5 <= A(3) AND B(5);
182 A4B5 <= A(4) AND B(5);
183 A5B5 <= A(5) AND B(5);
184 A6B5 <= A(6) AND B(5);
185 A7B5 <= A(7) AND B(5);
186
187 A0B6 <= A(0) AND B(6);
188 A1B6 <= A(1) AND B(6);
189 A2B6 <= A(2) AND B(6);
190 A3B6 <= A(3) AND B(6);
191 A4B6 <= A(4) AND B(6);
192 A5B6 <= A(5) AND B(6);
193 A6B6 <= A(6) AND B(6);
194 A7B6 <= A(7) AND B(6);
195
196 A0B7 <= A(0) AND B(7);
197 A1B7 <= A(1) AND B(7);
198 A2B7 <= A(2) AND B(7);
199 A3B7 <= A(3) AND B(7);
200 A4B7 <= A(4) AND B(7);
201 A5B7 <= A(5) AND B(7);
202 A6B7 <= A(6) AND B(7);
203 A7B7 <= A(7) AND B(7);
204

```

```

205  --first stage
206
207  P0 : P(0) <= A(0) AND B(0);
208
209  HA11 : Halfadder port map (A1B0,A0B1,P(1),C1);
210  FA12 : Fulladder port map (A2B0,A1B1,C1,S2,C2);
211  FA13 : Fulladder port map (A3B0,A2B1,C2,S3,C3);
212  FA14 : Fulladder port map (A4B0,A3B1,C3,S4,C4);
213  FA15 : Fulladder port map (A5B0,A4B1,C4,S5,C5);
214  FA16 : Fulladder port map (A6B0,A5B1,C5,S6,C6);
215  FA17 : Fulladder port map (A7B0,A6B1,C6,S7,C7);
216  HA18 : Halfadder port map (A7B1,C7,S8,C8);
217
218
219  --second stage
220
221  HA21 : Halfadder port map (S2,A0B2,P(2),C9);
222  FA22 : Fulladder port map (S3,A1B2,C9,S10,C10);
223  FA23 : Fulladder port map (S4,A2B2,C10,S11,C11);
224  FA24 : Fulladder port map (S5,A3B2,C11,S12,C12);
225  FA25 : Fulladder port map (S6,A4B2,C12,S13,C13);
226  FA26 : Fulladder port map (S7,A5B2,C13,S14,C14);
227  FA27 : Fulladder port map (S8,A6B2,C14,S15,C15);
228  FA28 : Fulladder port map (C8,A7B2,C15,S16,C16);
229
230  --third stage
231
232  HA31 : Halfadder port map (S10,A0B3,P(3),C17);
233  FA32 : Fulladder port map (S11,A1B3,C17,S18,C18);
234  FA33 : Fulladder port map (S12,A2B3,C18,S19,C19);
235  FA34 : Fulladder port map (S13,A3B3,C19,S20,C20);
236  FA35 : Fulladder port map (S14,A4B3,C20,S21,C21);
237  FA36 : Fulladder port map (S15,A5B3,C21,S22,C22);
238  FA37 : Fulladder port map (S16,A6B3,C22,S23,C23);
239  FA38 : Fulladder port map (C16,A7B3,C23,S24,C24);
240
241  --fourth stage
242
243  HA41 : Halfadder port map (S18,A0B4,P(4),C25);
244  FA42 : Fulladder port map (S19,A1B4,C25,S26,C26);
245  FA43 : Fulladder port map (S20,A2B4,C26,S27,C27);
246  FA44 : Fulladder port map (S21,A3B4,C27,S28,C28);
247  FA45 : Fulladder port map (S22,A4B4,C28,S29,C29);
248  FA46 : Fulladder port map (S23,A5B4,C29,S30,C30);
249  FA47 : Fulladder port map (S24,A6B4,C30,S31,C31);
250  FA48 : Fulladder port map (C24,A7B4,C31,S32,C32);
251
252  --fifth stage
253
254  HA51 : Halfadder port map (S26,A0B5,P(5),C33);
255  FA52 : Fulladder port map (S27,A1B5,C33,S34,C34);
256  FA53 : Fulladder port map (S28,A2B5,C34,S35,C35);
257  FA54 : Fulladder port map (S29,A3B5,C35,S36,C36);
258  FA55 : Fulladder port map (S30,A4B5,C36,S37,C37);
259  FA56 : Fulladder port map (S31,A5B5,C37,S38,C38);
260  FA57 : Fulladder port map (S32,A6B5,C38,S39,C39);
261  FA58 : Fulladder port map (C32,A7B5,C39,S40,C40);
262
263  --sixth stage
264
265  HA61 : Halfadder port map (S34,A0B6,P(6),C41);
266  FA62 : Fulladder port map (S35,A1B6,C41,S42,C42);
267  FA63 : Fulladder port map (S36,A2B6,C42,S43,C43);
268  FA64 : Fulladder port map (S37,A3B6,C43,S44,C44);
269  FA65 : Fulladder port map (S38,A4B6,C44,S45,C45);
270  FA66 : Fulladder port map (S39,A5B6,C45,S46,C46);
271  FA67 : Fulladder port map (S40,A6B6,C46,S47,C47);
272  FA68 : Fulladder port map (C40,A7B6,C47,S48,C48);
273
274  --seventh stage
275
276  HA71 : Halfadder port map (S42,A0B7,P(7),C49);
277  FA72 : Fulladder port map (S43,A1B7,C49,P(8),C50);
278  FA73 : Fulladder port map (S44,A2B7,C50,P(9),C51);
279  FA74 : Fulladder port map (S45,A3B7,C51,P(10),C52);
280  FA75 : Fulladder port map (S46,A4B7,C52,P(11),C53);
281  FA76 : Fulladder port map (S47,A5B7,C53,P(12),C54);
282  FA77 : Fulladder port map (S48,A6B7,C54,P(13),C55);
283  FA78 : Fulladder port map (A7B7,C48,C55,P(14),P(15));
284
285  end Behavioral;

```

Appendix 2.0

Wallace Multiplier Program

```
20 :
21 :
22 : library IEEE;
23 : use IEEE.STD_LOGIC_1164.ALL;
24 :
25 : -- Uncomment the following library declaration if using
26 : -- arithmetic functions with Signed or Unsigned values
27 : --use IEEE.NUMERIC_STD.ALL;
28 :
29 : -- Uncomment the following library declaration if instantiating
30 : -- any Xilinx leaf cells in this code.
31 : --library UNISIM;
32 : --use UNISIM.VComponents.all;
33 :
34 : entity Wallace is
35 :     Port ( A : in STD_LOGIC_Vector(7 downto 0);
36 :           B : in STD_LOGIC_Vector(7 downto 0);
37 :           F : out STD_LOGIC_Vector(15 downto 0));
38 : end Wallace;
39 :
40 : architecture Behavioral of Wallace is
41 :
42 :     component Fulladder is
43 :         Port ( A : in STD_Logic;
44 :               B : in STD_Logic;
45 :               Cin : in STD_Logic;
46 :               Sum : out STD_Logic;
47 :               Cout : out STD_Logic);
48 :     end component;
49 :
50 :     component Halfadder is
51 :         Port ( A : in STD_LOGIC;
52 :               B : in STD_LOGIC;
53 :               Sum : out STD_LOGIC;
54 :               Cout : out STD_LOGIC);
55 :     end component;
56 :
57 : 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;
58 : 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,C26,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;
59 :
```

```

60 signal A1B0 : STD_Logic;
61 signal A2B0 : STD_Logic;
62 signal A3B0 : STD_Logic;
63 signal A4B0 : STD_Logic;
64 signal A5B0 : STD_Logic;
65 signal A6B0 : STD_Logic;
66 signal A7B0 : STD_Logic;
67
68 signal A0B1 : STD_Logic;
69 signal A1B1 : STD_Logic;
70 signal A2B1 : STD_Logic;
71 signal A3B1 : STD_Logic;
72 signal A4B1 : STD_Logic;
73 signal A5B1 : STD_Logic;
74 signal A6B1 : STD_Logic;
75 signal A7B1 : STD_Logic;
76
77 signal A0B2 : STD_Logic;
78 signal A1B2 : STD_Logic;
79 signal A2B2 : STD_Logic;
80 signal A3B2 : STD_Logic;
81 signal A4B2 : STD_Logic;
82 signal A5B2 : STD_Logic;
83 signal A6B2 : STD_Logic;
84 signal A7B2 : STD_Logic;
85
86 signal A0B3 : STD_Logic;
87 signal A1B3 : STD_Logic;
88 signal A2B3 : STD_Logic;
89 signal A3B3 : STD_Logic;
90 signal A4B3 : STD_Logic;
91 signal A5B3 : STD_Logic;
92 signal A6B3 : STD_Logic;
93 signal A7B3 : STD_Logic;
94
95 signal A0B4 : STD_Logic;
96 signal A1B4 : STD_Logic;
97 signal A2B4 : STD_Logic;
98 signal A3B4 : STD_Logic;
99 signal A4B4 : STD_Logic;
100 signal A5B4 : STD_Logic;
101 signal A6B4 : STD_Logic;
102 signal A7B4 : STD_Logic;
103
104 signal A0B5 : STD_Logic;
105 signal A1B5 : STD_Logic;
106 signal A2B5 : STD_Logic;
107 signal A3B5 : STD_Logic;
108 signal A4B5 : STD_Logic;
109 signal A5B5 : STD_Logic;
110 signal A6B5 : STD_Logic;
111 signal A7B5 : STD_Logic;
112
113 signal A0B6 : STD_Logic;
114 signal A1B6 : STD_Logic;
115 signal A2B6 : STD_Logic;
116 signal A3B6 : STD_Logic;
117 signal A4B6 : STD_Logic;
118 signal A5B6 : STD_Logic;
119 signal A6B6 : STD_Logic;
120 signal A7B6 : STD_Logic;
121
122 signal A0B7 : STD_Logic;
123 signal A1B7 : STD_Logic;
124 signal A2B7 : STD_Logic;
125 signal A3B7 : STD_Logic;
126 signal A4B7 : STD_Logic;
127 signal A5B7 : STD_Logic;
128 signal A6B7 : STD_Logic;
129 signal A7B7 : STD_Logic;
130

```

```

131 begin
132
133 A1B0 <= A(1) AND B(0);
134 A2B0 <= A(2) AND B(0);
135 A3B0 <= A(3) AND B(0);
136 A4B0 <= A(4) AND B(0);
137 A5B0 <= A(5) AND B(0);
138 A6B0 <= A(6) AND B(0);
139 A7B0 <= A(7) AND B(0);
140
141 A0B1 <= A(0) AND B(1);
142 A1B1 <= A(1) AND B(1);
143 A2B1 <= A(2) AND B(1);
144 A3B1 <= A(3) AND B(1);
145 A4B1 <= A(4) AND B(1);
146 A5B1 <= A(5) AND B(1);
147 A6B1 <= A(6) AND B(1);
148 A7B1 <= A(7) AND B(1);
149
150 A0B2 <= A(0) AND B(2);
151 A1B2 <= A(1) AND B(2);
152 A2B2 <= A(2) AND B(2);
153 A3B2 <= A(3) AND B(2);
154 A4B2 <= A(4) AND B(2);
155 A5B2 <= A(5) AND B(2);
156 A6B2 <= A(6) AND B(2);
157 A7B2 <= A(7) AND B(2);
158
159 A0B3 <= A(0) AND B(3);
160 A1B3 <= A(1) AND B(3);
161 A2B3 <= A(2) AND B(3);
162 A3B3 <= A(3) AND B(3);
163 A4B3 <= A(4) AND B(3);
164 A5B3 <= A(5) AND B(3);
165 A6B3 <= A(6) AND B(3);
166 A7B3 <= A(7) AND B(3);
167
168 A0B4 <= A(0) AND B(4);
169 A1B4 <= A(1) AND B(4);
170 A2B4 <= A(2) AND B(4);
171 A3B4 <= A(3) AND B(4);
172 A4B4 <= A(4) AND B(4);
173 A5B4 <= A(5) AND B(4);
174 A6B4 <= A(6) AND B(4);
175 A7B4 <= A(7) AND B(4);
176
177 A0B5 <= A(0) AND B(5);
178 A1B5 <= A(1) AND B(5);
179 A2B5 <= A(2) AND B(5);
180 A3B5 <= A(3) AND B(5);
181 A4B5 <= A(4) AND B(5);
182 A5B5 <= A(5) AND B(5);
183 A6B5 <= A(6) AND B(5);
184 A7B5 <= A(7) AND B(5);
185
186 A0B6 <= A(0) AND B(6);
187 A1B6 <= A(1) AND B(6);
188 A2B6 <= A(2) AND B(6);
189 A3B6 <= A(3) AND B(6);
190 A4B6 <= A(4) AND B(6);
191 A5B6 <= A(5) AND B(6);
192 A6B6 <= A(6) AND B(6);
193 A7B6 <= A(7) AND B(6);
194
195 A0B7 <= A(0) AND B(7);
196 A1B7 <= A(1) AND B(7);
197 A2B7 <= A(2) AND B(7);
198 A3B7 <= A(3) AND B(7);
199 A4B7 <= A(4) AND B(7);
200 A5B7 <= A(5) AND B(7);
201 A6B7 <= A(6) AND B(7);
202 A7B7 <= A(7) AND B(7);
203

```

```

204 --first stage
205
206 P0 : P(0) <= A(0) AND B(0);
207
208 HA11 : Halfadder port map (A1B0,A0B1,P(1),C1);
209 FA12 : Fulladder port map (A2B0,A1B1,A0B2,S2,C2);
210 FA13 : Fulladder port map (A3B0,A2B1,A1B2,S3,C3);
211 FA14 : Fulladder port map (A4B0,A3B1,A2B2,S4,C4);
212 FA15 : Fulladder port map (A5B0,A4B1,A3B2,S5,C5);
213 FA16 : Fulladder port map (A6B0,A5B1,A4B2,S6,C6);
214 FA17 : Fulladder port map (A7B0,A6B1,A5B2,S7,C7);
215 HA18 : Halfadder port map (A7B1,A6B2,S8,C8);
216
217
218 --second stage
219
220 HA21 : Halfadder port map (S2,C1,P(2),C9);
221 FA22 : Fulladder port map (S3,C2,A0B3,S10,C10);
222 FA23 : Fulladder port map (S4,C3,A1B3,S11,C11);
223 FA24 : Fulladder port map (S5,C4,A2B3,S12,C12);
224 FA25 : Fulladder port map (S6,C5,A3B3,S13,C13);
225 FA26 : Fulladder port map (S7,C6,A4B3,S14,C14);
226 FA27 : Fulladder port map (S8,C7,A5B3,S15,C15);
227 FA28 : Fulladder port map (C8,A7B2,A6B3,S16,C16);
228
229 --third stage
230
231 HA31 : Halfadder port map (S10,C9,P(3),C17);
232 FA32 : Fulladder port map (S11,C10,A0B4,S18,C18);
233 FA33 : Fulladder port map (S12,C11,A1B4,S19,C19);
234 FA34 : Fulladder port map (S13,C12,A2B4,S20,C20);
235 FA35 : Fulladder port map (S14,C13,A3B4,S21,C21);
236 FA36 : Fulladder port map (S15,C14,A4B4,S22,C22);
237 FA37 : Fulladder port map (S16,C15,A5B4,S23,C23);
238 FA38 : Fulladder port map (C16,A7B3,A6B4,S24,C24);
239
240 --fourth stage
241
242 HA41 : Halfadder port map (S18,C17,P(4),C25);
243 FA42 : Fulladder port map (S19,C18,A0B5,S26,C26);
244 FA43 : Fulladder port map (S20,C19,A1B5,S27,C27);
245 FA44 : Fulladder port map (S21,C20,A2B5,S28,C28);
246 FA45 : Fulladder port map (S22,C21,A3B5,S29,C29);
247 FA46 : Fulladder port map (S23,C22,A4B5,S30,C30);
248 FA47 : Fulladder port map (S24,C23,A5B5,S31,C31);
249 FA48 : Fulladder port map (C24,A7B4,A6B5,S32,C32);
250
251 --fifth stage
252
253 HA51 : Halfadder port map (S26,C25,P(5),C33);
254 FA52 : Fulladder port map (S27,C26,A0B6,S34,C34);
255 FA53 : Fulladder port map (S28,C27,A1B6,S35,C35);
256 FA54 : Fulladder port map (S29,C28,A2B6,S36,C36);
257 FA55 : Fulladder port map (S30,C29,A3B6,S37,C37);
258 FA56 : Fulladder port map (S31,C30,A4B6,S38,C38);
259 FA57 : Fulladder port map (S32,C31,A5B6,S39,C39);
260 FA58 : Fulladder port map (C32,A7B5,A6B6,S40,C40);
261

```

```

262 --sixth stage
263
264 HA61 : Halfadder port map (S34,C33,P(6),C41);
265 FA62 : Fulladder port map (S35,C34,A0B7,S42,C42);
266 FA63 : Fulladder port map (S36,C35,A1B7,S43,C43);
267 FA64 : Fulladder port map (S37,C36,A2B7,S44,C44);
268 FA65 : Fulladder port map (S38,C37,A3B7,S45,C45);
269 FA66 : Fulladder port map (S39,C38,A4B7,S46,C46);
270 FA67 : Fulladder port map (S40,C39,A5B7,S47,C47);
271 FA68 : Fulladder port map (C40,A7B6,A6B7,S48,C48);
272
273 --seventh stage
274
275 HA71 : Halfadder port map (S42,C41,P(7),C49);
276 FA72 : Fulladder port map (S43,C42,C49,P(8),C50);
277 FA73 : Fulladder port map (S44,C43,C50,P(9),C51);
278 FA74 : Fulladder port map (S45,C44,C51,P(10),C52);
279 FA75 : Fulladder port map (S46,C45,C52,P(11),C53);
280 FA76 : Fulladder port map (S47,C46,C53,P(12),C54);
281 FA77 : Fulladder port map (S48,C47,C54,P(13),C55);
282 FA78 : Fulladder port map (A7B7,C48,C55,P(14),P(15));
283
284 end Behavioral;

```