

ACDA6001 Electronics 2

By Alex Gravatt (2217176)

Contents

Table of Contents

Contents	2
Abstract	5
Introduction	5
Design 6	
logic circuit design	6
Multiplier block diagrams	8
Implementation	10
RTL design	11
constraints	12
I/O planning	13
Implementation schematic	15
device	16
hardware implementation	22
Testing	24
Evaluation	33
design and implementation	33
timing	33
power consumption	33
speed and power efficiency	33
LUTs	33
fpga implementation	34
tests	34
tools and techniques	34
Conclusion	34
Bibliography	34
Appendix 1	35
Appendix 2	45
Appendix 3	54

Figures

Table of Figures

Figure 1: half adder circuit	7
Figure 2: full adder circuit	9

Figure 3: braun dot diagram [11]	11
Figure 4: Braun multiplier logic diagram [2]	12
Figure 5: Wallace dot diagram[10]	13
Figure 6: Wallace logic diagram	14
Figure 7: Braun RTL	15
Figure 8: Wallace RTL	16
Figure 9: Braun multiplier longest path [2]	17
Figure 10: Wallace multiplier longest path	18
Figure 11: io planning Braun	19
Figure 12: io planning Wallace	20
Figure 13: implementation diagram Braun	21
Figure 14: implementation diagram Wallace	21
Figure 15: device implantation Braun	22
Figure 16: device implementation Wallace	23
Figure 17: Braun LUT usage	23
Figure 18: Braun pin out 1	24
Figure 19: Braun pin out 2	24
Figure 20: timing Braun synthesised	24
Figure 21: Braun utilisation timing and power implemented	25
Figure 22: Braun timing implemented	25
Figure 23: Braun power 1	26
Figure 24: Braun power 2	26
Figure 25: Wallace LUT usage	26
Figure 26: Wallace pin-out 1	27
Figure 27: Wallace pin-out 2	27
Figure 28: Wallace timing synthesised	27
Figure 29: Wallace power utilisation and timing implemented	28
Figure 30: Wallace power	28
Figure 31: Braun basys test calculator [5]	29
Figure 32: Braun on basys3 board test	29
Figure 33: Wallace basys3 test calculator [5]	30
Figure 34: Wallace basys3 test	30
Figure 35: Braun SIM test 1	31
Figure 36: Braun SIM test Calc 1[5]	32
Figure 37: Braun SIM test 2	32
Figure 38: Braun SIM test Calc 2 [5]	33
Figure 39: Braun SIM test 3	33
Figure 40: Braun SIM test Calc 3 [5]	34
Figure 41: Braun SIM test 4	34
Figure 42: Braun SIM Calc 4 [5]	35
Figure 43: Wallace SIM test 1	35
Figure 44: Wallace Calc test 1 [5]	36
Figure 45: Wallace sim test2	36
Figure 46: Wallace SIM Calc 2 [5]	37
Figure 47: Wallace SIM test 3	38
Figure 48: Wallace SIM Calc 3 [5]	38
Figure 49: Wallace SIM test 4	39
Figure 50: Wallace SIM Calc 4[5]	39
Figure 51: Braun code 1	44

Figure 52: Braun code 2	46
Figure 53: Braun code 3	48
Figure 54: Braun code 4	50
Figure 55: Braun code 5	52
Figure 56: Braun code 5	53
Figure 57: Braun code 6	55
Figure 58: Braun code 7	57
Figure 59: Braun code 8	59
Figure 60: Wallace code 1	61
Figure 61: Wallace code 2	62
Figure 62: Wallace code 3	64
Figure 63: Wallace code 4	66
Figure 64: Wallace code 5	67
Figure 65: Wallace code 6	68
Figure 66: Wallace code 7	69
Figure 67: Wallace code 8	70
Figure 68: Wallace code 9	71
Figure 69: Wallace code 10	72
Figure 70: constants code 1	73
Figure 71: constants code 2	74

1 Abstract

In this report the author demonstrates Wallace and Braun multiplier developed in Vivado. The author also demonstrates the similarities and difference of the Braun and Wallace multiplier. The results of this report show that Wallace multiples are preferred in any case where there is no need for partial computation as it is faster smaller and more power efficient than Braun.

2 Introduction

This report will demonstrate how to design, implement and simulate two binary multipliers using Vivado, an integrated development environment produced by Xilinx now owned by AMD. these multipliers will be of 8×8 size and simulated on a basys3 development board

binary multipliers are regularly used in micro controllers, computers and FPGA's anywhere where the user will need to perform multiplication calculations. From the earliest computers capable of multiplication such as the “mark-1 from the USA”[7] the mark1 was developed in 1943 when Wallace multipliers were not developed as such this machine likely used Bruan multipliers. Computers capable of multiplication needed a form of multiplier. There are two focused on in this report Braun and Wallace. Braun and Wallace multipliers take unsigned number inputs and produce a multiplied output.

The Braun multiplier was first implemented in a “mechanical calculator made by Anton Braun” [8] these are the basis of modern electronic Braun binary multipliers.

The Wallace multipliers were first devised by Chris Wallace in 1964 when he was asked to provide suggestions on high speed multiplication methods. His methods used operand reduction in parallel layers. [9]

Literature Review

This literature review examines two texts that inform on VHDL design “Fundamentals Of Digital Logic With VHDL Design 3rd Edition” [5] and “free range VHDL” [6].

“Fundamentals Of Digital Logic With VHDL Design 3rd Edition” [5]

This book provides a from the ground up approach to how to design logic circuits in VHDL.

This book goes through the design basics, the implementation technologies and numeric representation of arithmetic circuits.

The author of this book provides a good foundation for novices. This was useful for the author of this report. They provide numerous snippets of VHDL code as well as touching on multipliers. The multipliers section assisted the author of this report with checking their designs.

in comparison with a book like free range VHDL it provides a more circuit based look at VHDL design and focuses more on the circuits rather than the VHDL code that free VHDL looks at.

The author of the book has a from basics up approach with a start at basic logic and logic gates working its way up to sequential circuits. This approach benefits learners but will be less efficient to those with strong fundamentals such as experts in the field.

“free range VHDL” [6]

Free range VHDL is a book focused on the VHDL and how to best design VHDL code such as its syntax applications and best practices. This book helped the author best understand how to layout their VHDL code according to best practices and how to prepare for the design process.

Unlike fundamentals of digital logic with VHDL" changed to "This contrasts with Brown and Vranesic's (2008) 'Fundamentals of Digital Logic with VHDL Design,' which adopts a more circuit-centric approach.

3 Design

4 In this section the author designs the block diagrams for the Wallace and Braun multiplier and the logic circuits.

logic circuit design

half adder logic

The half adder is a logic circuit comprised of an and an or logic gate that produces a sum and a carry from two inputs circuit below.

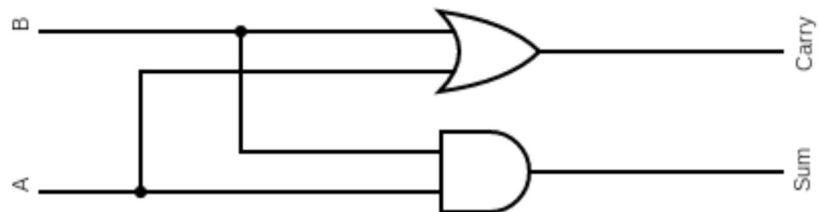


Figure 1: half adder circuit

the half adder has a truth table demonstrated below

A	B	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

you can simplify the truth table into two k maps shown below
sum

ba	0	1
0	0	1

1	1	0
---	---	---

(A or B)= sum

carry

ba	0	1
0	0	0
1	0	1

(A and B)=carry

full adder logic

the full adder logic circuit comprising the gates shown below.

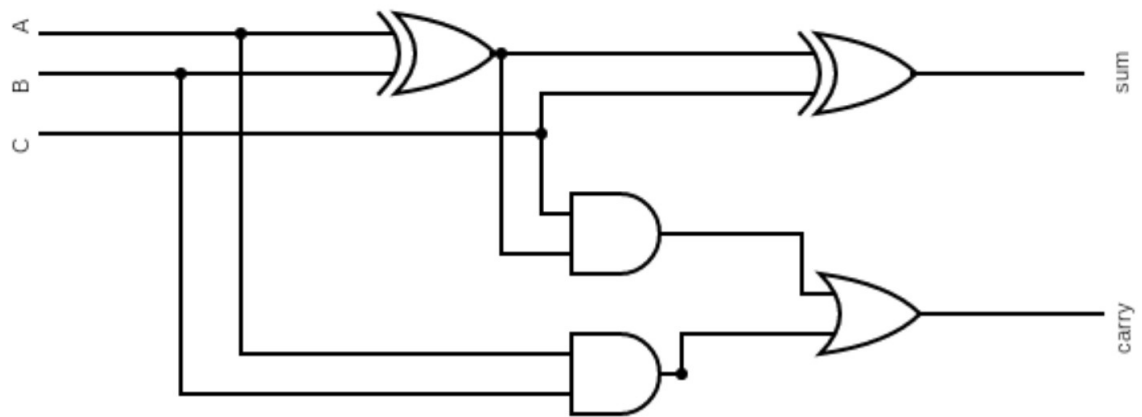


Figure 2: full adder circuit

It produces two outputs: a carry and a sum from three inputs.
the truth table is as below

a	b	c	sum	carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1

1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The full adder truth table can be simplified into the two maps below

abc	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$\text{sum} = (A \text{ XOR } B \text{ XOR } c)$$

carry

abc	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$\text{carry} = AB + AC + BC$$

After identifying the outputs and logic design of the half and full adder the author designed them in VHDL to reduce the time taken later these were to be defined as components of the system.

Multiplier block diagrams

The author drew the multipliers out ahead of producing the VHDL code for the multiplier to allow them to better visualise the paths the circuit would take and how the VHDL would need to be written.

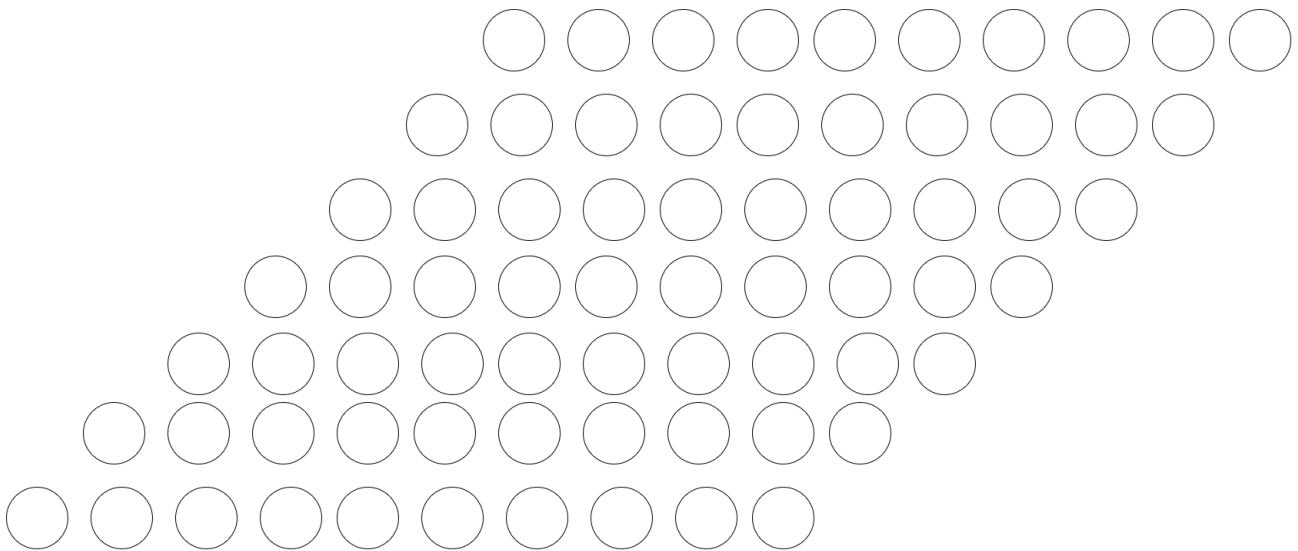


Figure 3: braun dot diagram [11]

The dot diagram demonstrates the structure of the multiplier diagram

The 8×8 Braun multiplier is as shown below

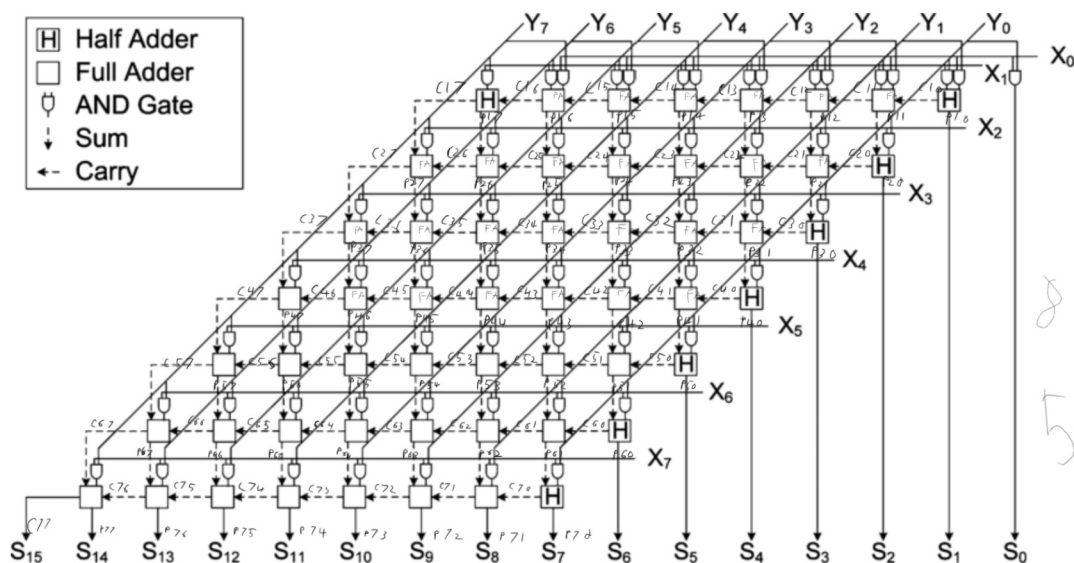


Figure 4: Braun multiplier logic diagram [2]

In the authors 8×8 Braun multiplier they map out all the internal signals that the latter implementations will use to reduce development time.

The Wallace multiplier is drawn similarly; the only difference is that the drawing used as a base by the author combined the last set of adders into a single block called vector merging that was broken out into its respective adders by the author as shown below.

Wallace dot diagram Demonstrates how the multiplier is constructed.

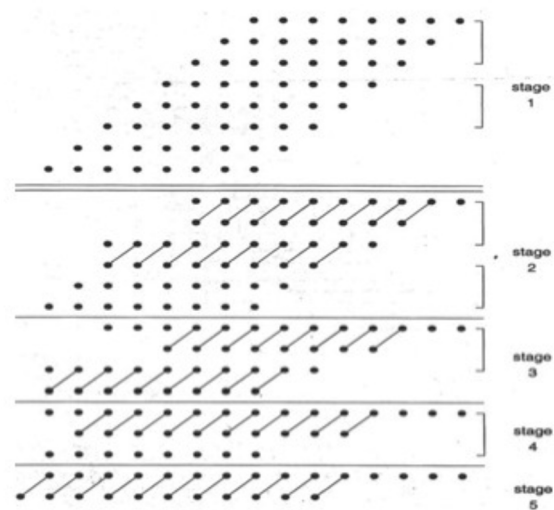


Figure 5: Wallace dot diagram[10]

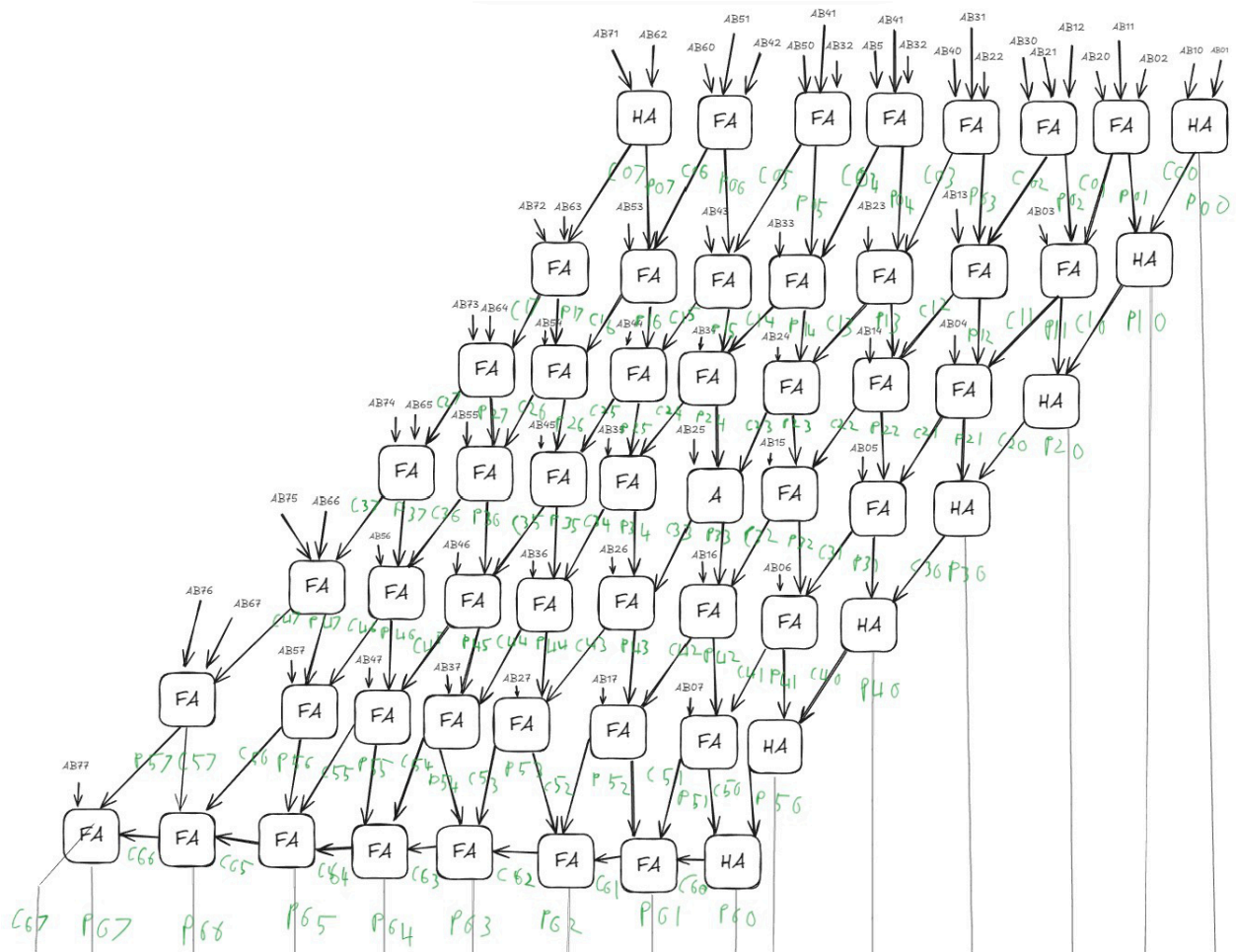


Figure 6: Wallace logic diagram

Implementation

Using the drawings made in the design the author produced an 8×8 Braun multiplier the variables are named P, C, S, AB these decided on based on the names on the drawing. The P is the internal output of the sums from each half and full adder with the first number after the P being the row or bank of the adder and the second being where it is in the bank.

C follows the same format as P only that C is for Carries.

S is the final sum and only appears at the bottom of the multiplier.

AB are the outputs of the Ands taken from the A, B input with the first number being what bank and the second being where in the bank it is located AB0 is bank 0.

These are defined as vectors giving them 0 to 7 and 15 in the case of the 8×8 this allows the AB to iterate through the A parts in the multiplier.

Mapping as vectors reduces the number of logic statements the author has to make and allows for easier coding. This allows the author to reference different logic's outputs using `c1(0)` this means in the authors code carry bank1 array value 1.

The Wallace multiplier the author designed is defined in the same way as the Braun multiplier allowing for the use of the same inputs and outputs in the constraints file.

RTL design

Figure 7: Braun RTL

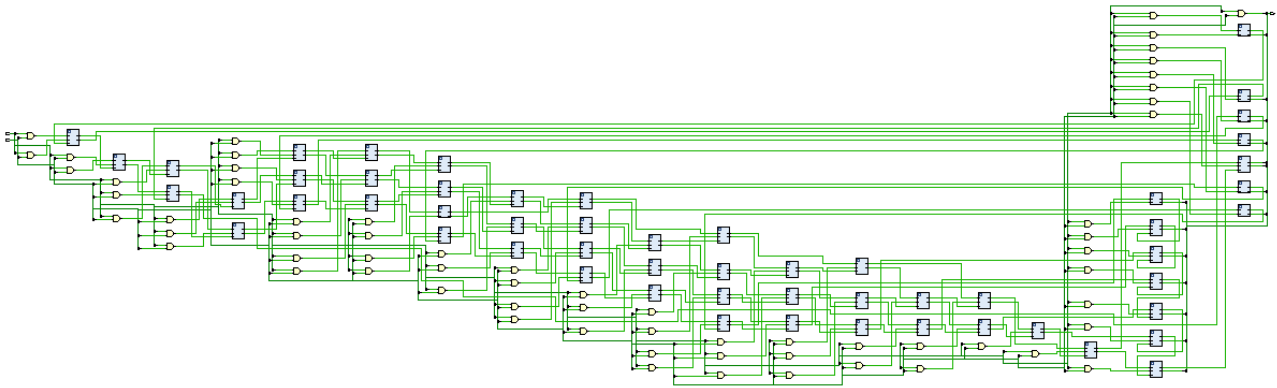


Figure 7: Braun RTL

Wallace

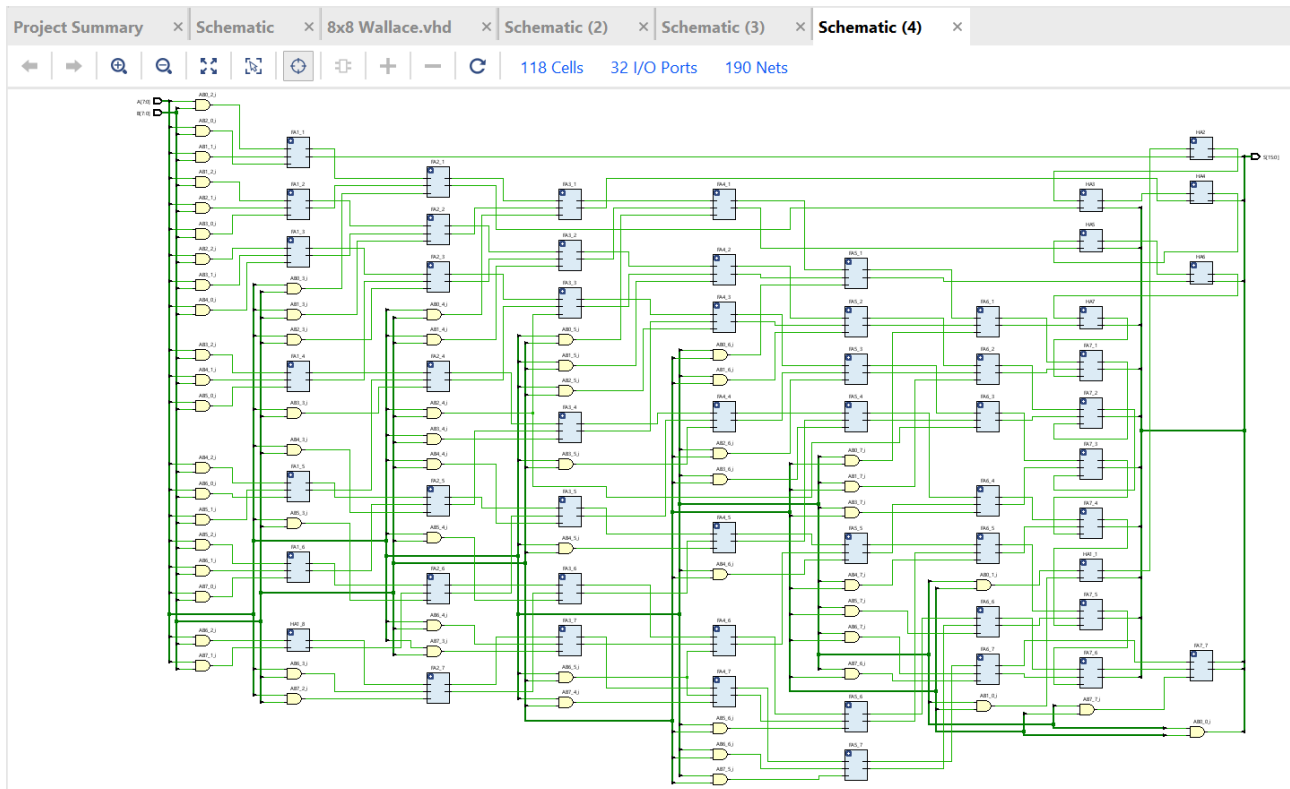


Figure 8: Wallace RTL

constraints

for a half adder the delay is 1 nano second.

for a full adder the delay is 2 nanoseconds

For the constraints file the author defined the minimum input delay for both the A port and B ports; they also defined the max input delay for both. The author defines the max as 1 microsecond as this is the longest it should take for an input to be registered on the author's process. This is different from the max time to input which would cause errors if there was no input at all.

the author defined the virtual clock on the constraints to have a 20 microsecond period

The author defined the output of S port to have a max of 2 micro second and a min of 0.

The author then defined the minimum delay from port A to Port B For Braun to 0 and the max as 47 microseconds; this is the max length the process should take to get through the longest route. The author calculated the Wallace multiplier as 0 to 26 as it passes through 13 full adders.

Code in appendix 3

Below is how the author calculated the maximum length for both Wallace and Braun.

Braun

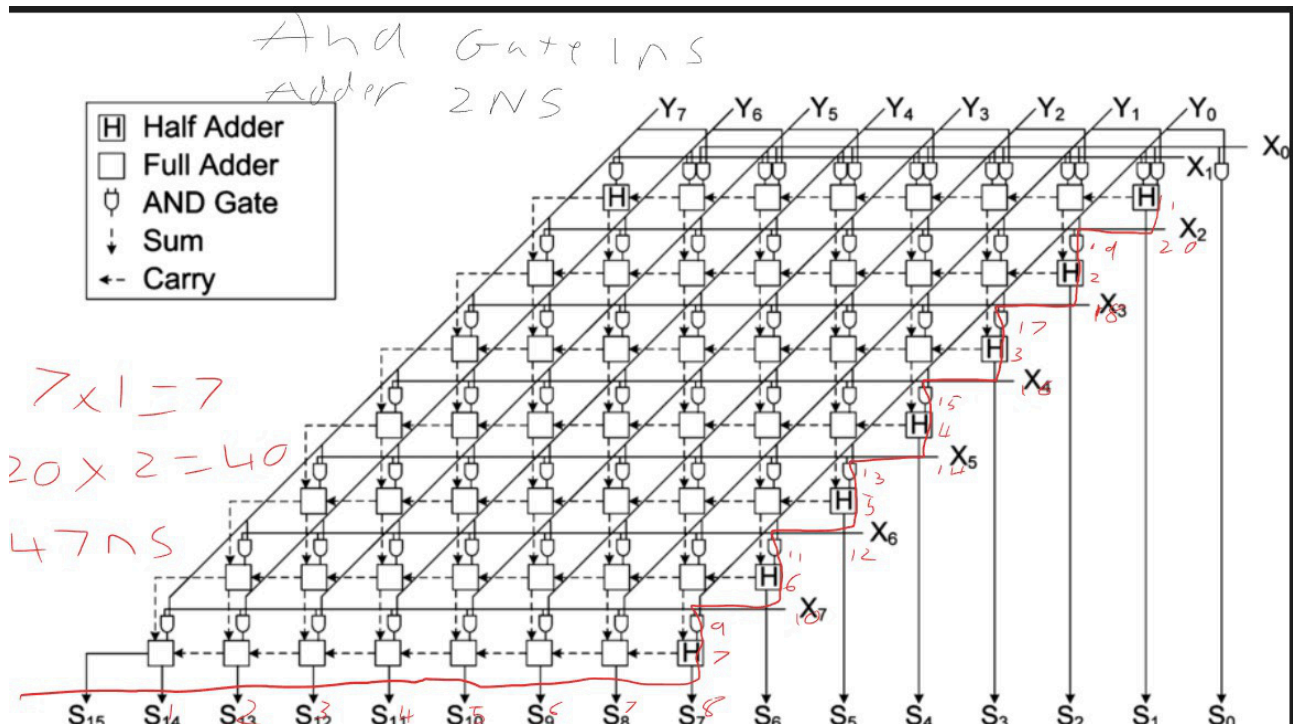


Figure 9: Braun multiplier longest path [2]

Wallace

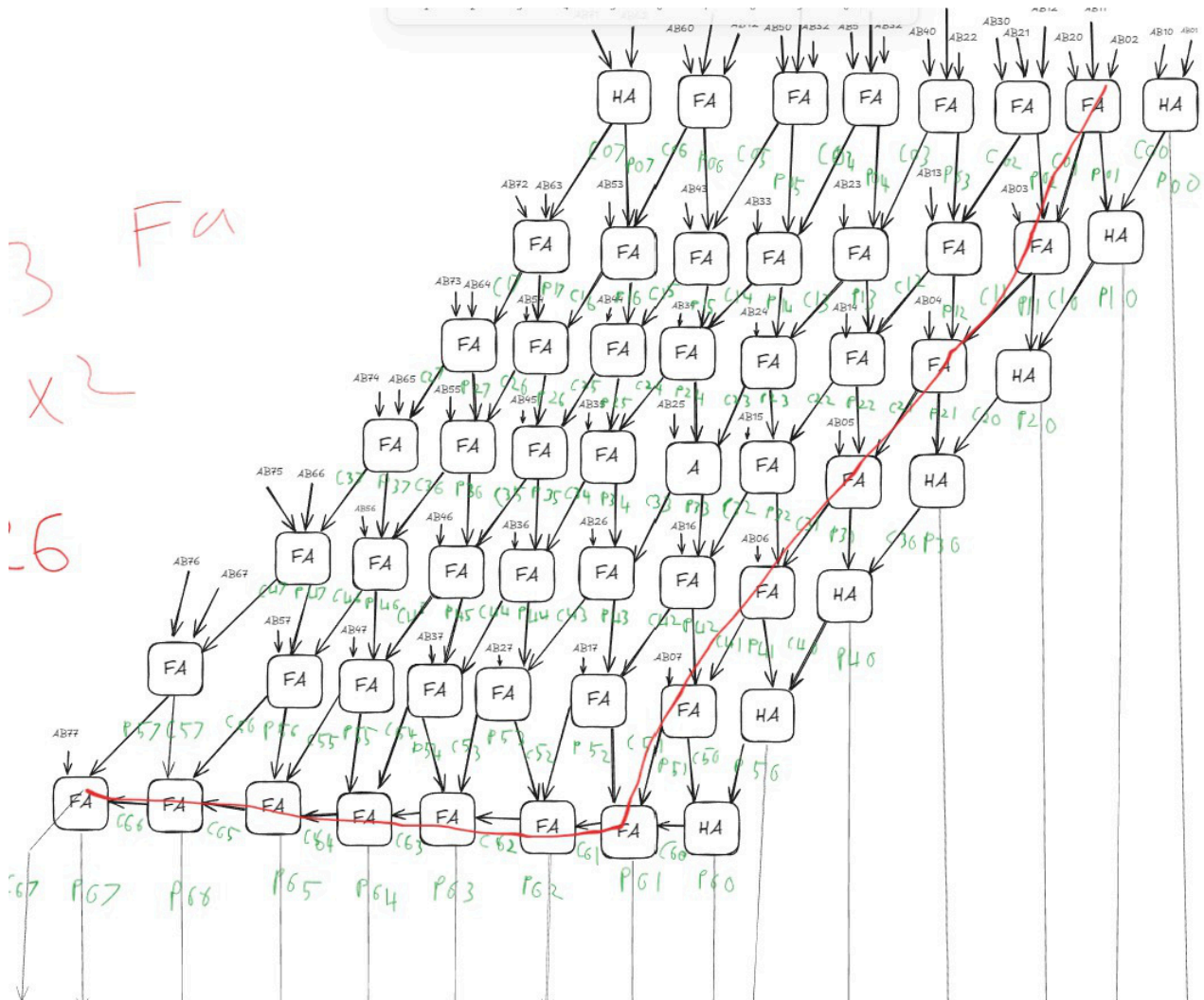


Figure 10: Wallace multiplier longest path

I/O planning

The author defined pins A1 to B2 as the inputs and B3 to C4 as the outputs these were then set to have their standard for IO set as LVCMOS18 this is a standard that works at 1.8 V this decision was made to minimise power usage and to follow the advice of the program that the user should not leave the program to default. This advice is given to the author to allow the author to look at the next component in the circuit and check what it needs as an output. In the case of this program it will be powering a simulation or a led these are not power intensive parts.

LVCMOS comes in many versions the most common are 33 25 18 the difference is listed in the sysIO document[3] from lattice Corporation they indicate that the main differences are slew rate max voltage and amperage with 33 maxing at 24mA and 18 maxing at 12mA[3].

Braun



Figure 11: io planning Braun

Wallace



Figure 12: io planning Wallace

Implementation schematic

Braun

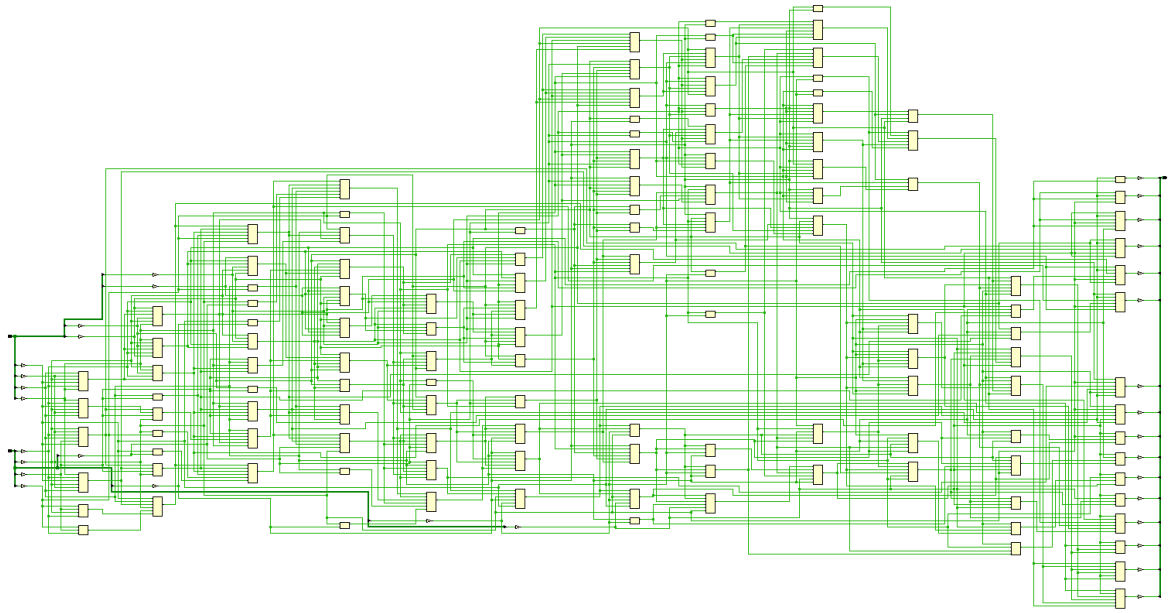


Figure 13: implementation diagram Braun

Wallace

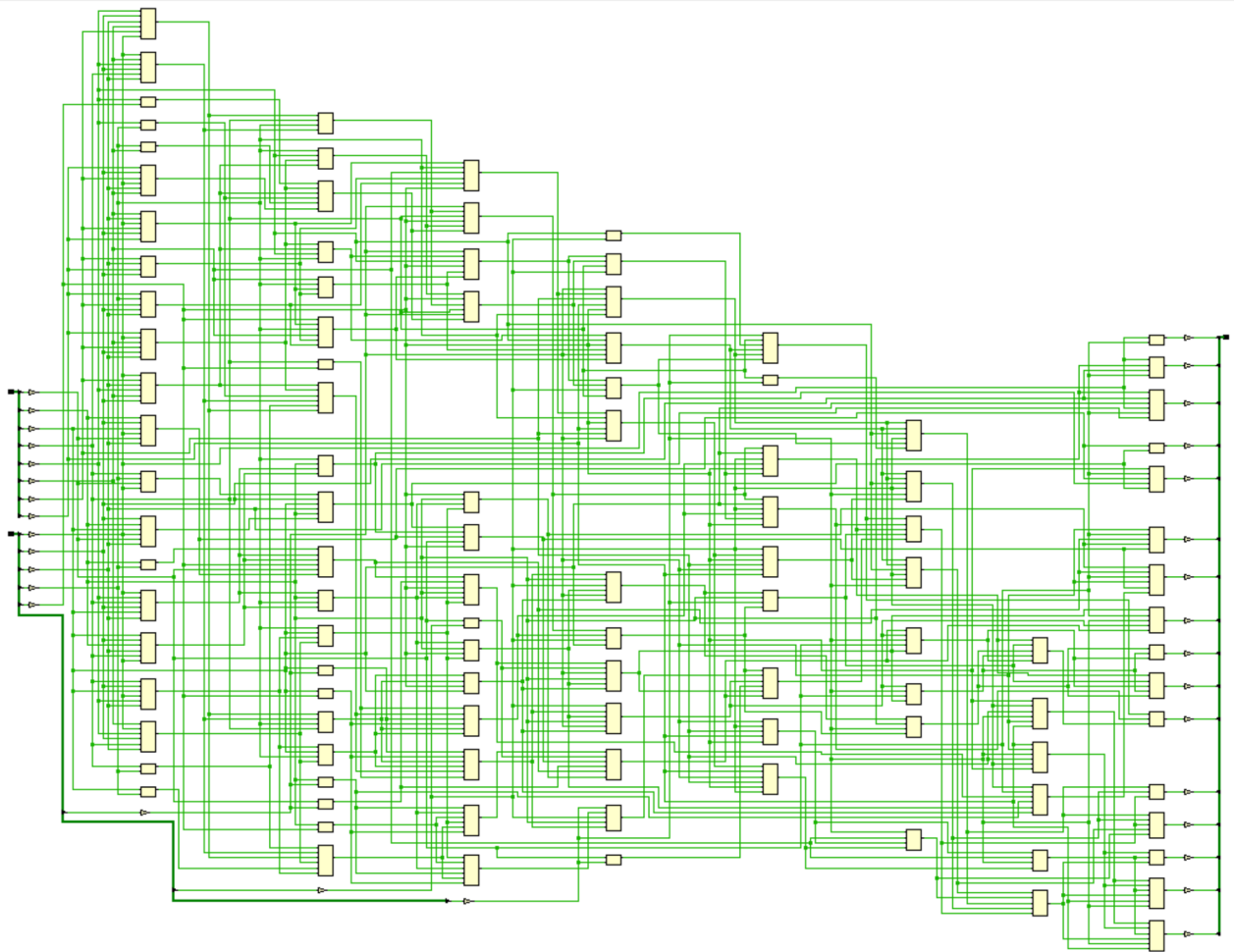


Figure 14: implementation diagram Wallace

These schematics are generated by the Vivado design software to make it easier to trace how the logic of the circuit is flowing. This particular schematic is more difficult to read than the RTL design view as it is how the circuit is implemented on FPGA rather than in concept.

device

Braun

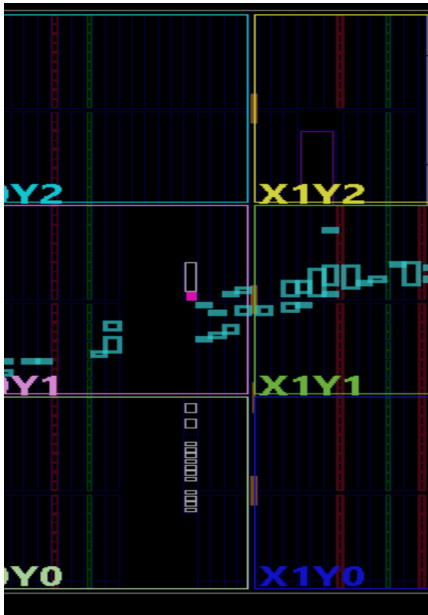


Figure 15: device implantation

Braun

Wallace

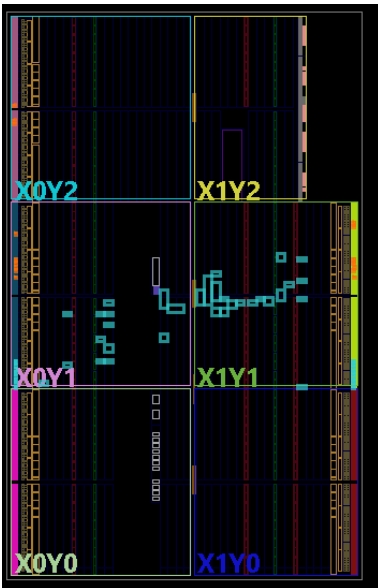


Figure 16: device
implementation Wallace

This shows how the circuit is allocated across the FPGA from pin to logic bit this is not something the user controls in Vivado as the software looks to avoid the user causing inefficiency inside the circuit.

Below is the setup of both Wallace and Braun in their constraints and design.

Braun

LUT'S used

Reports	Design Runs	Utilization	×	Power	Timing	Package Pins
Q ≡ ≡ % Hierarchy						
Name		Slice LUTs (10400)	Slice (8150)	LUT as Logic (10400)	Bonded IOB (210)	
N braunmulti8x8		105	45	105	32	

Figure 17: Braun LUT usage

pin setup

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
▼ All ports (32)													
▼ A (8)	IN			✓	(Multiple)	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ A[7]	IN		A1	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ A[6]	IN		A3	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ A[5]	IN		A4	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ A[4]	IN		A5	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ A[3]	IN		A6	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ A[2]	IN		A8	✓	16	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ A[1]	IN		A9	✓	16	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ A[0]	IN		A10	✓	16	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ B (8)	IN			✓	(Multiple)	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ B[7]	IN		A11	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ B[6]	IN		A13	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ B[5]	IN		A14	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ B[4]	IN		A15	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ B[3]	IN		A16	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ B[2]	IN		A18	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ B[1]	IN		B1	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▶ B[0]	IN		B2	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼

Figure 18: Braun pin out 1

▼ S (16)	OUT			✓	(Multiple)	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[15]	OUT		B3	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[14]	OUT		B4	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[13]	OUT		B6	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[12]	OUT		B7	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[11]	OUT		B8	✓	16	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[10]	OUT		B9	✓	16	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[9]	OUT		B11	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[8]	OUT		B12	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[7]	OUT		B13	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[6]	OUT		B14	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[5]	OUT		B16	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[4]	OUT		B17	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[3]	OUT		B18	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[2]	OUT		C1	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[1]	OUT		C2	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▶ S[0]	OUT		C4	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼

Figure 19: Braun pin out 2

timing

Timing			
Design Timing Summary			
General Information			
Timer Settings			
Design Timing Summary			
Methodology Summary			
> Check Timing (0)			
Intra-Clock Paths			
Inter-Clock Paths			
> Other Path Groups			
User Ignored Paths			
Timing Summary - timing_1			
Setup	Hold	Pulse Width	
Worst Negative Slack (WNS): 32.518 ns	Worst Hold Slack (WHS): 1.709 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 20: timing Braun synthesised

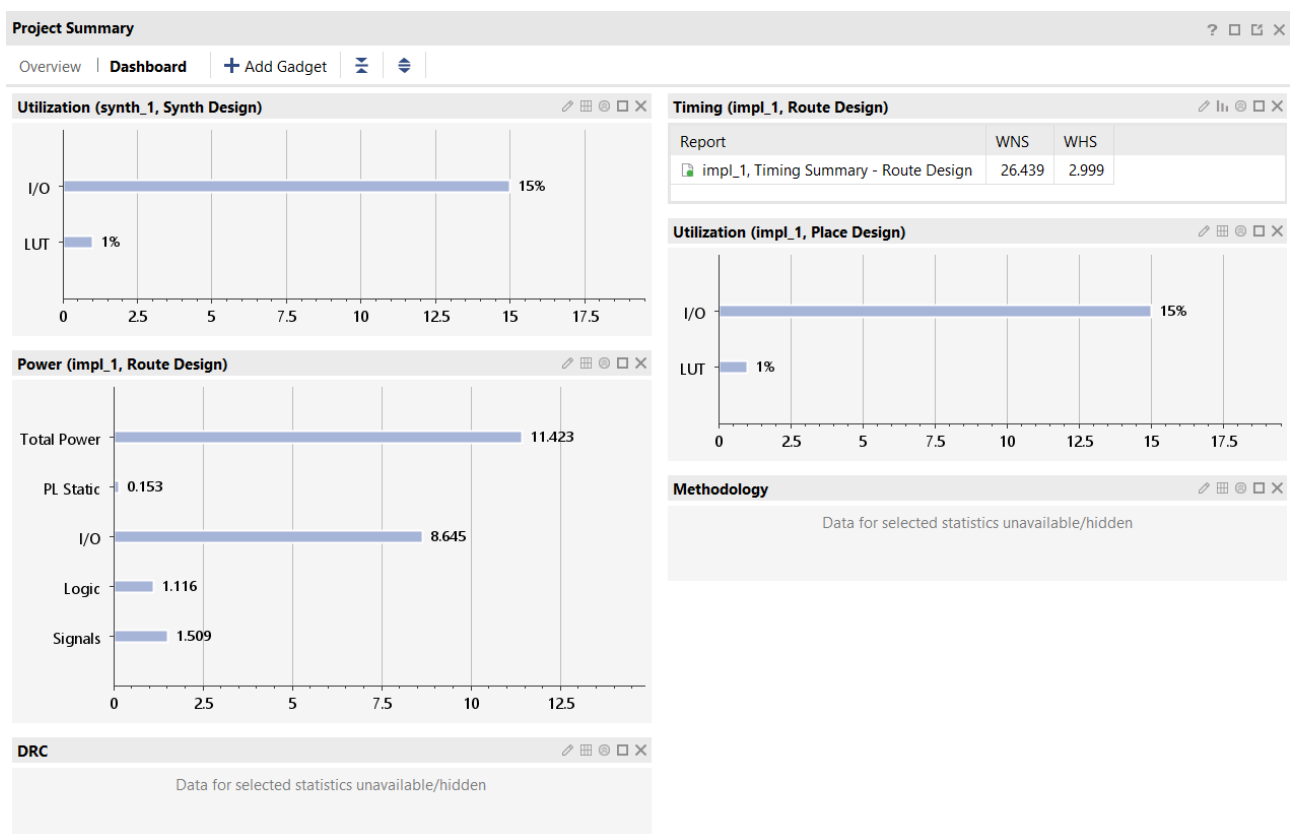


Figure 21: Braun utilisation timing and power implemented

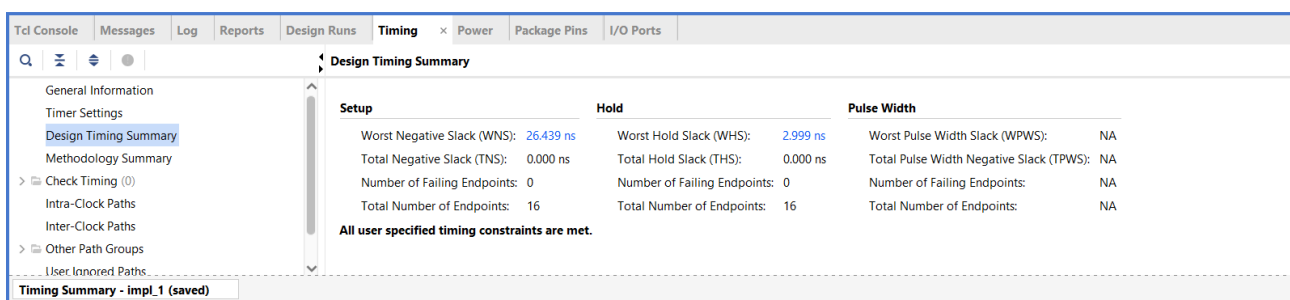


Figure 22: Braun timing implemented

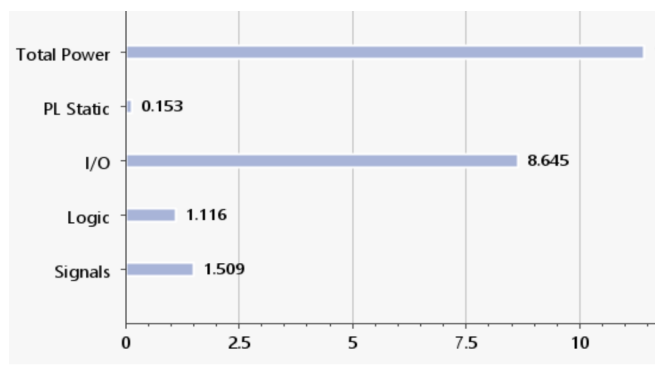


Figure 23: Braun power 1

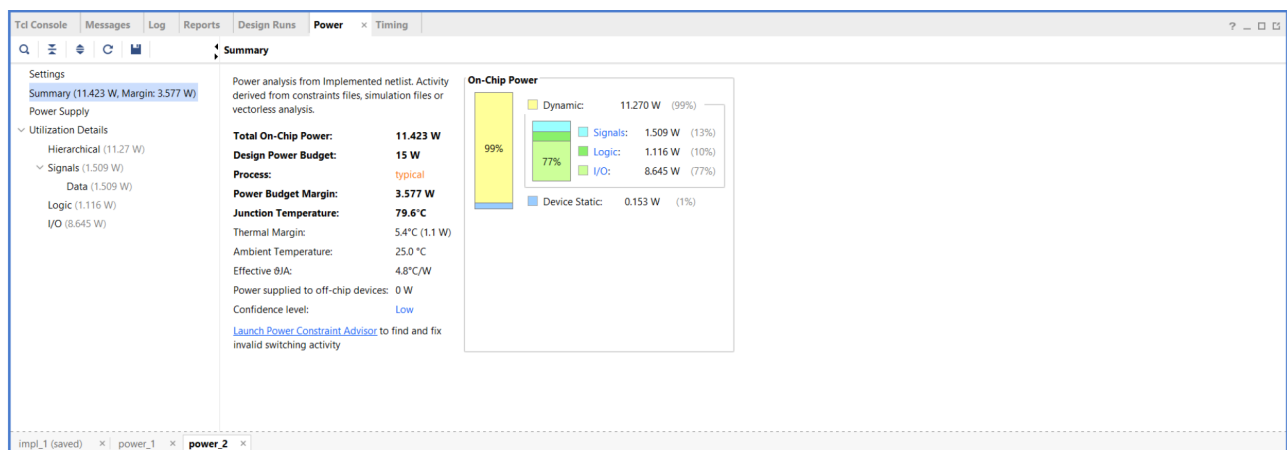


Figure 24: Braun power 2

Wallace

LUT'S used

Name	Slice LUTs (10400)	Bonded IOB (210)
Wallace_multiplier8bit	81	32

Figure 25: Wallace LUT's usage

pin setup

Tcl Console Messages Log Reports Design Runs I/O Ports x Power													
Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM
▼ All ports (32)													
▼ A (8)	IN			✓	(Multiple)	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ A[7]	IN		A1	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ A[6]	IN		A3	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ A[5]	IN		A4	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ A[4]	IN		A5	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ A[3]	IN		A6	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ A[2]	IN		A8	✓	16	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ A[1]	IN		A9	✓	16	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ A[0]	IN		A10	✓	16	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ B (8)	IN			✓	(Multiple)	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ B[7]	IN		A11	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ B[6]	IN		A13	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ B[5]	IN		A14	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ B[4]	IN		A15	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ B[3]	IN		A16	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ B[2]	IN		A18	✓	15	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ B[1]	IN		B1	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼
▼ B[0]	IN		B2	✓	35	LVC MOS15*	1.500				NONE	▼ NONE	▼

Figure 26: Wallace pin-out 1

▼ S (16)	OUT			✓	(Multiple)	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[15]	OUT		B3	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[14]	OUT		B4	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[13]	OUT		B6	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[12]	OUT		B7	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[11]	OUT		B8	✓	16	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[10]	OUT		B9	✓	16	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[9]	OUT		B11	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[8]	OUT		B12	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[7]	OUT		B13	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[6]	OUT		B14	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[5]	OUT		B16	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[4]	OUT		B17	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[3]	OUT		B18	✓	15	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[2]	OUT		C1	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[1]	OUT		C2	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼
▼ S[0]	OUT		C4	✓	35	LVC MOS15*	1.500	12	▼	▼	NONE	▼ FP_VTT_50	▼

Figure 27: Wallace pin-out 2

timing

Tcl Console Messages Log Reports Design Runs Timing x													
Design Timing Summary													
General Information													
Timer Settings													
Design Timing Summary													
Methodology Summary													
Check Timing (0)													
Intra-Clock Paths													
Inter-Clock Paths													
Other Path Groups													
User Ignored Paths													
Timing Summary - timing 1													
Setup													
Hold													
Pulse Width													
Worst Negative Slack (WNS): 14.730 ns													
Worst Hold Slack (WHS): 1.706 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 28: Wallace timing synthesised

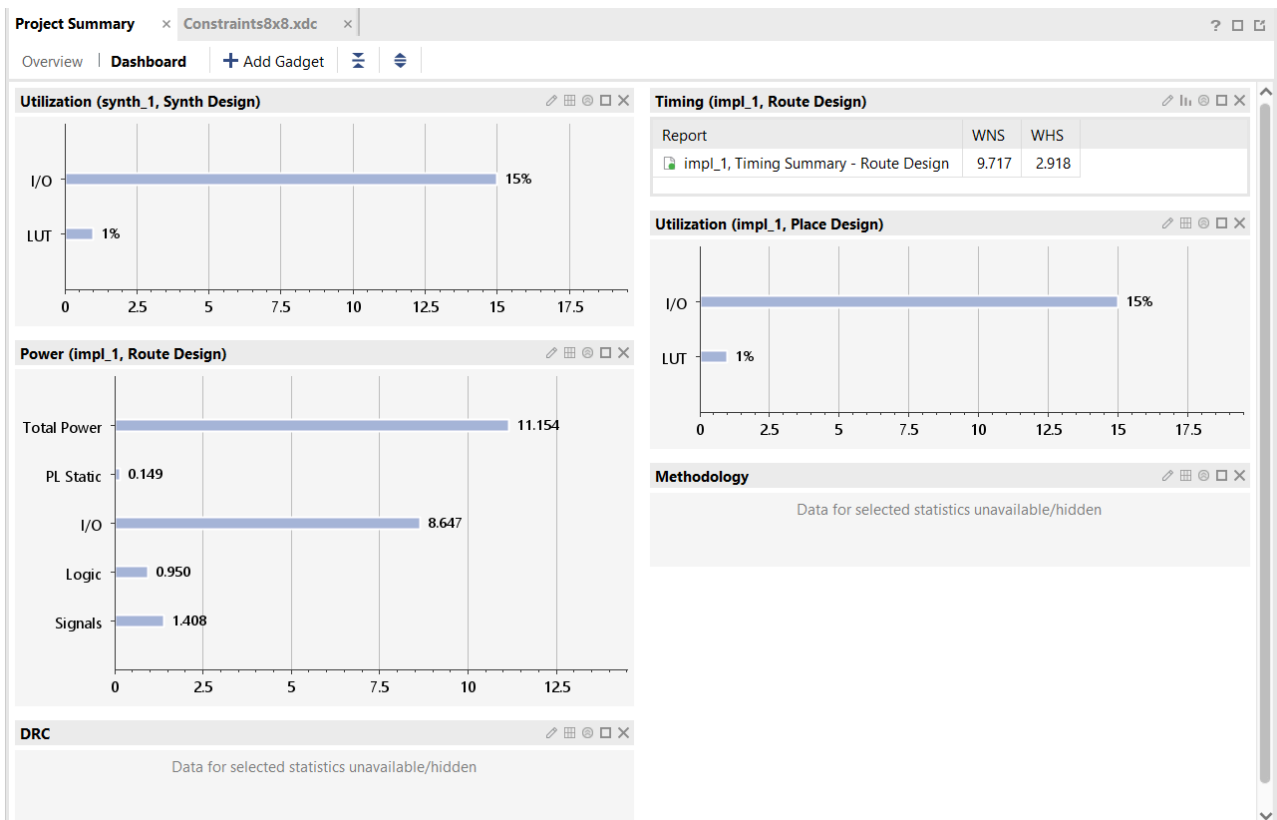


Figure 29: Wallace power utilisation and timing implemented

power

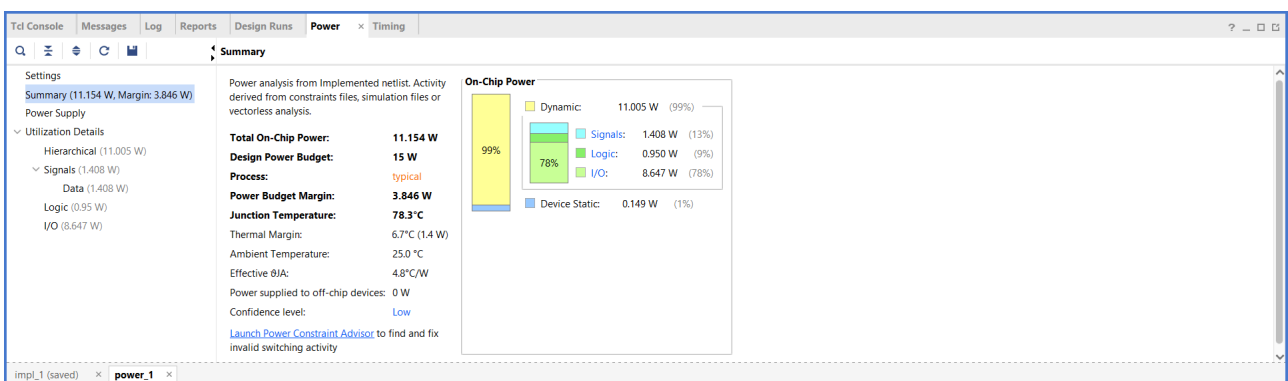


Figure 30: Wallace power

hardware implementation

Braun

$$10101010 \times 11001100 \\ = 01000011101111000$$

Decimal value:

$$170 \times 204$$

$$= 34680$$

10101010	×	11001100	= ?
Calculate		Clear	

Figure 31: Braun basys test calculator [5]

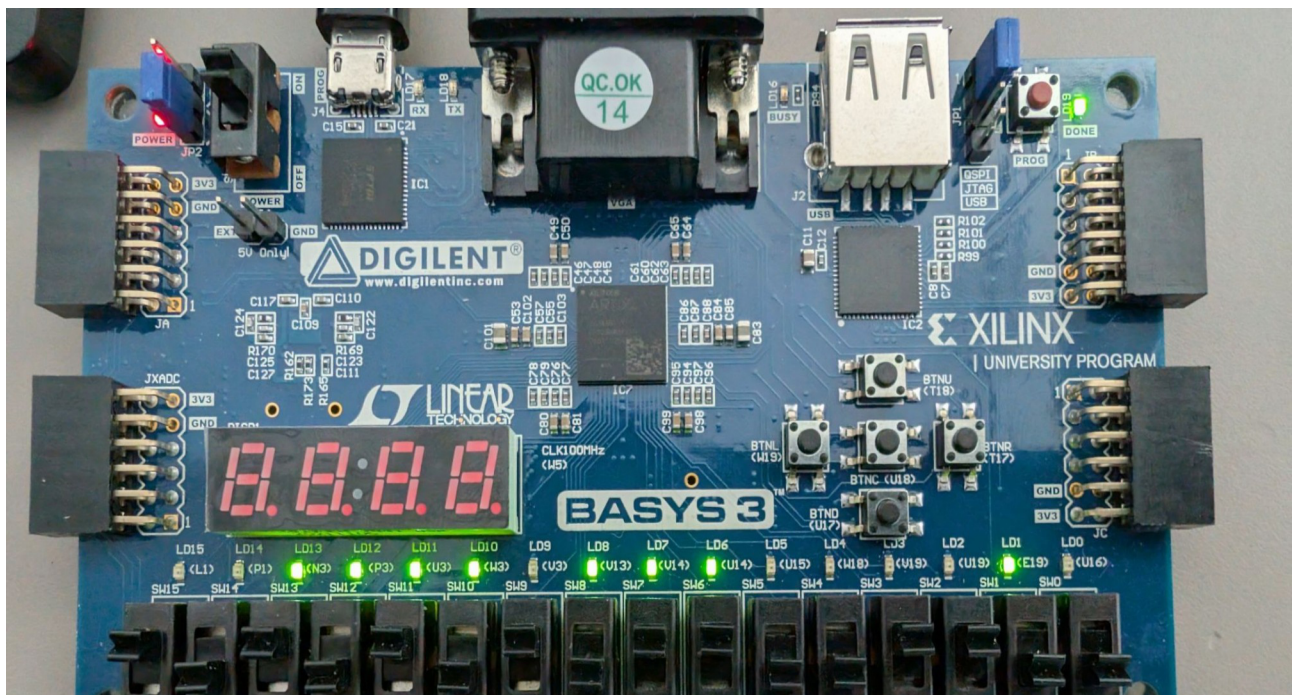


Figure 32: Braun on basys3 board test

read right to left on the led's

an and b inputs are swapped between v2 w13

Wallace works the same this is the program working below

Binary Calculator

Binary Calculation—Add, Subtract, Multiply, or Divide

Result



Binary value:

00110011×11110000
= **01011111010000**

Decimal value:

51×240
= **12240**

00110011	×	11110000	= ?
Calculate Clear			

Figure 33: Wallace basys3 test calculator [5]

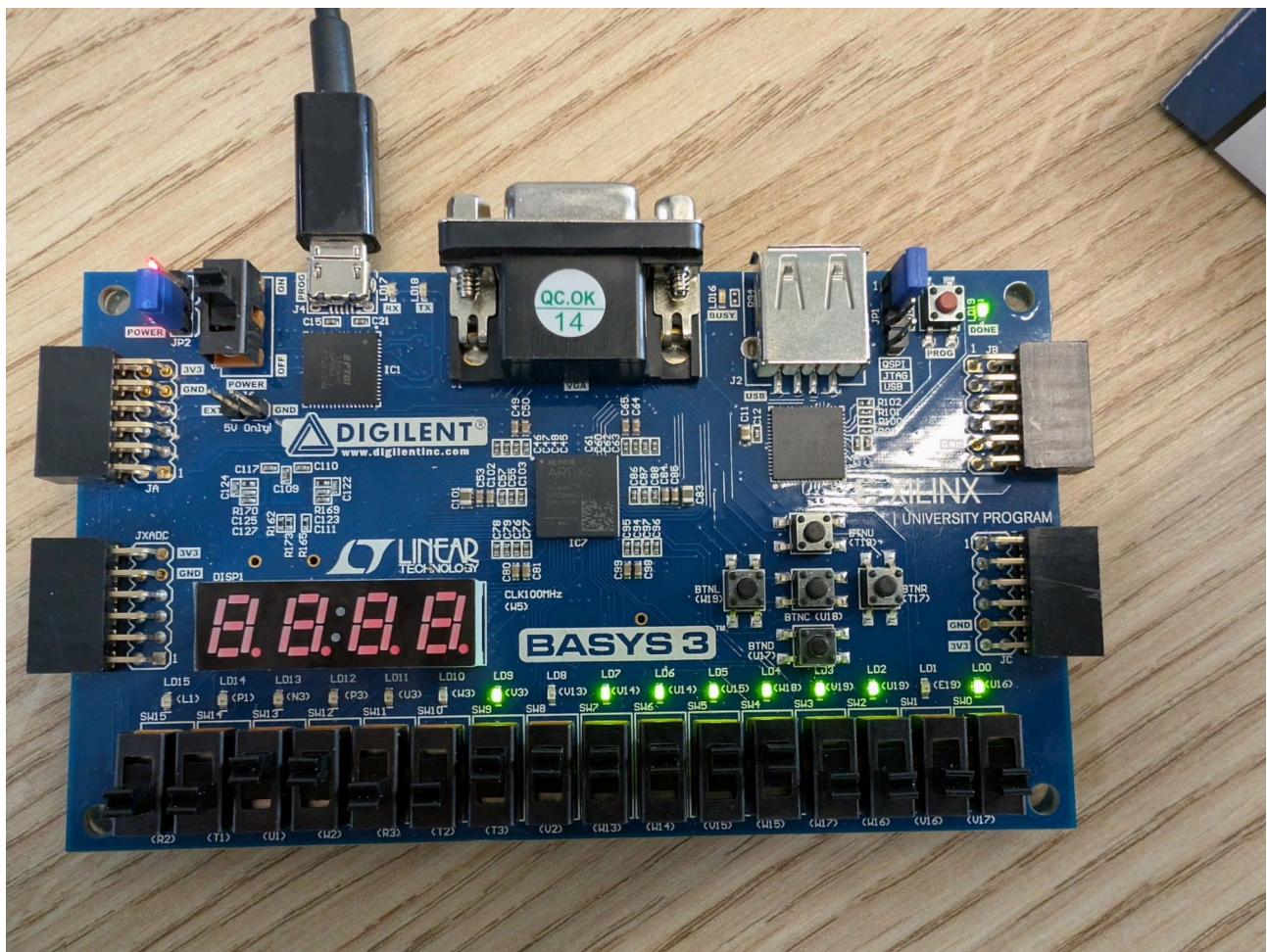


Figure 34: Wallace basys3 test

all code is in appendix 1

6 Testing

Braun

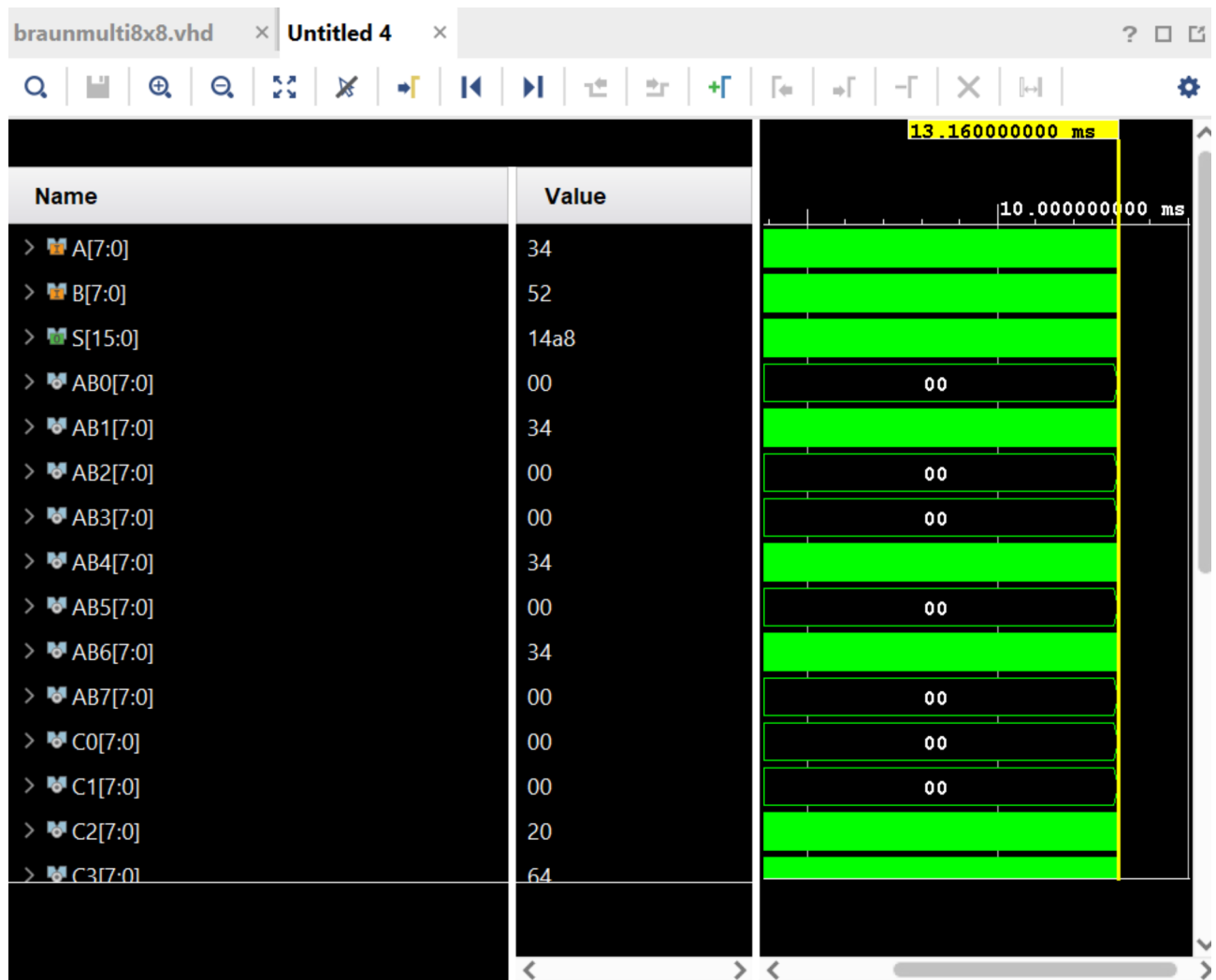


Figure 35: Braun SIM test 1

34=52

52=82

14a8=4264

Result save

Hex value:
 $34 \times 52 = 10A8$

Decimal value:
 $52 \times 82 = 4264$

= ?

Calculate

Clear

Figure 36: Braun SIM test Calc 1[5]

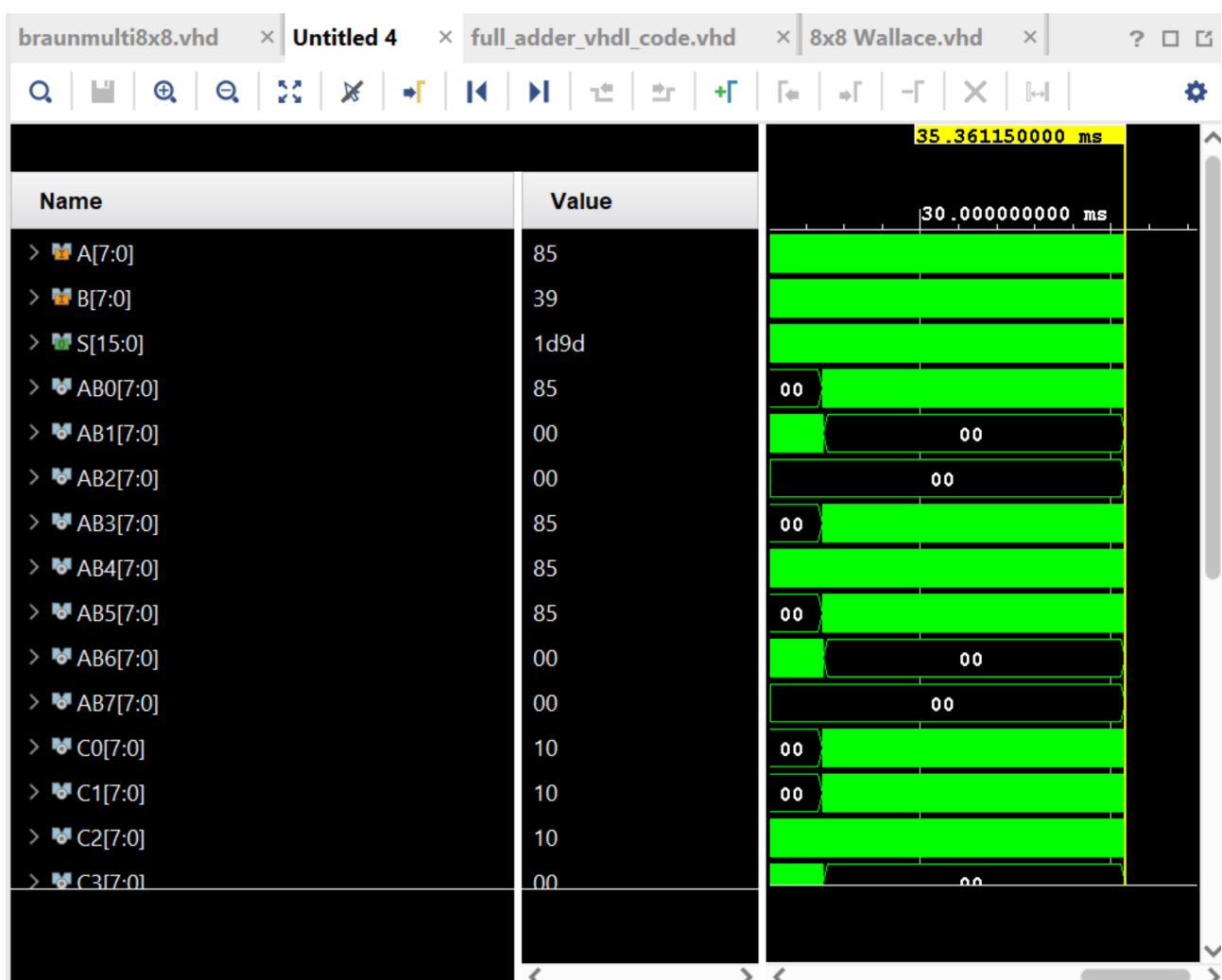


Figure 37: Braun SIM test 2

85=133
39=57
1d9d=7581

Result save

Hex value:

$85 \times 39 = 1D9D$

Decimal value:

$133 \times 57 = 7581$

85	×	▼	39	= ?
Calculate Clear				

Figure 38: Braun SIM test Calc 2 [5]

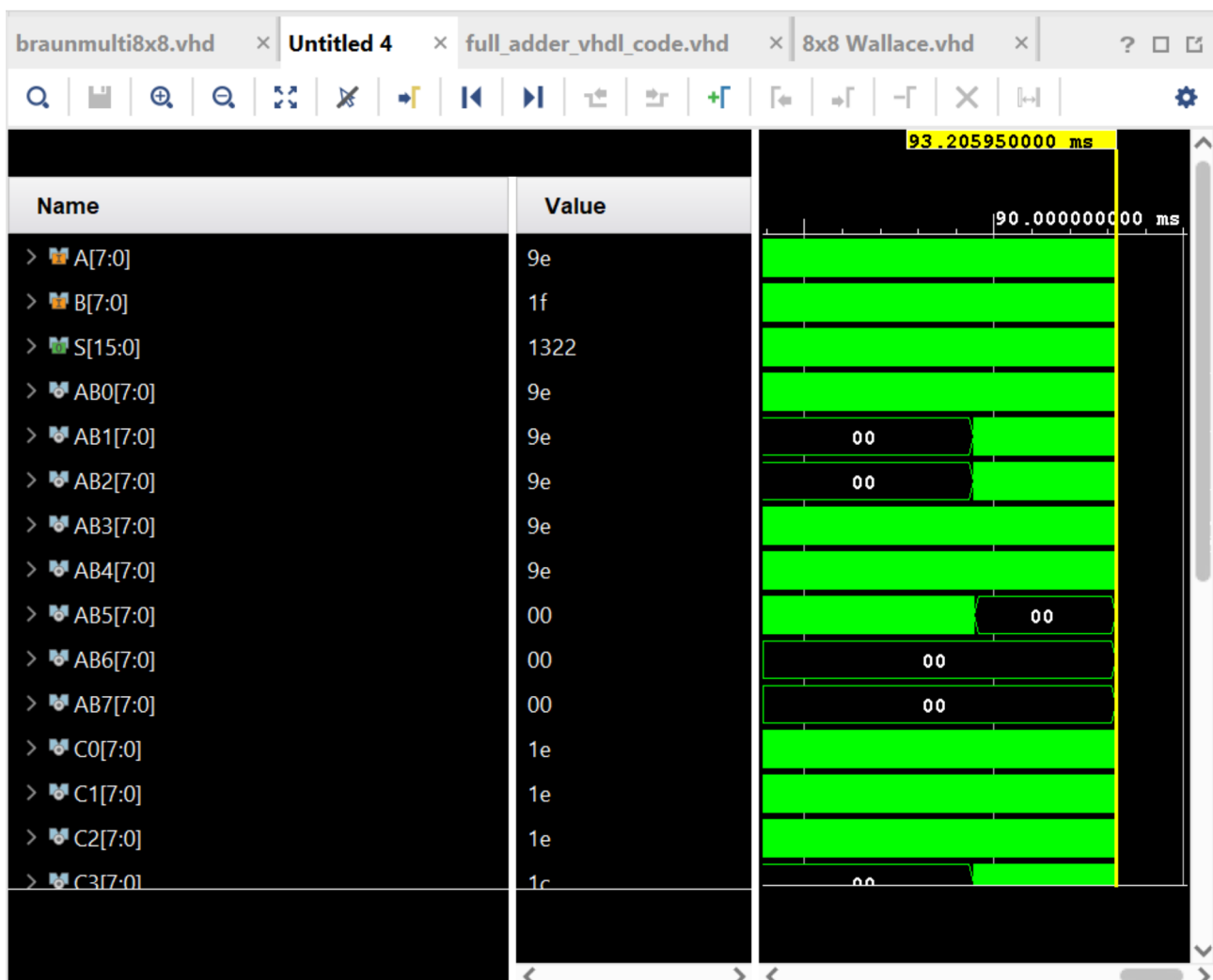


Figure 39: Braun SIM test 3

9e = 158

1f = 31

1322 = 4898

Hex value:
9e × 1f = 1322

Decimal value:
158 × 31 = 4898

9e

×

1f

= ?

Calculate

Clear

Figure 40: Braun SIM test Calc 3 [5]

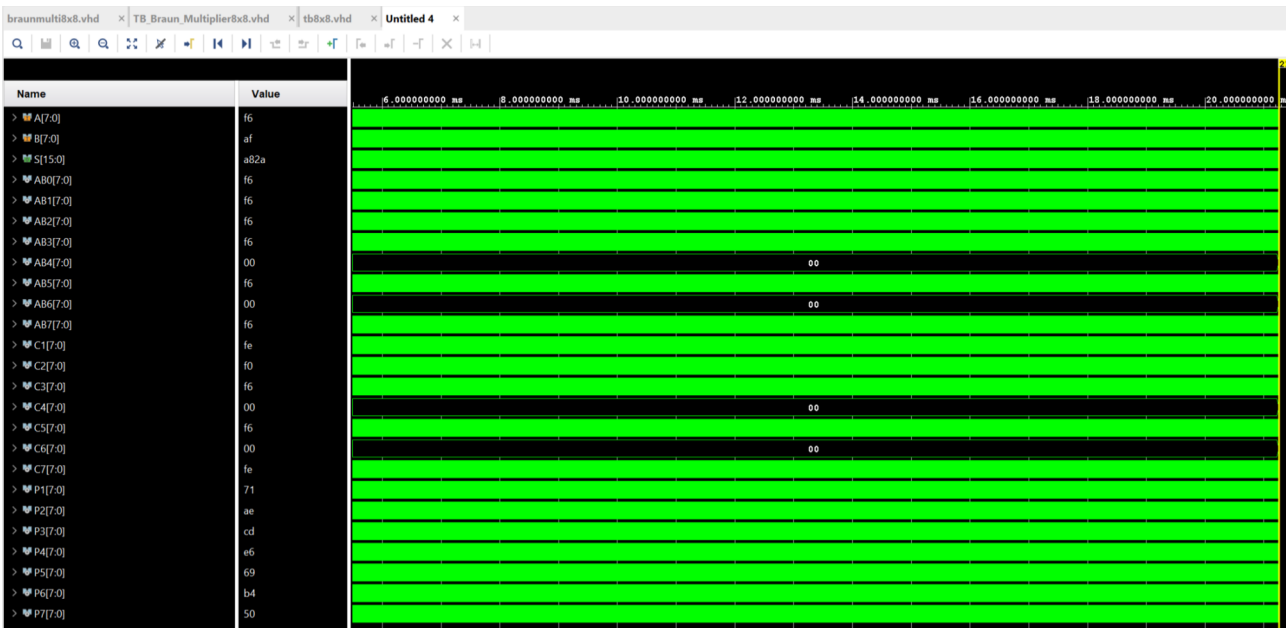


Figure 41: Braun SIM test 4

f6=246
fa=175
a82a= 43050

Result save

Hex value:

$f6 \times af = \mathbf{A82A}$

Decimal value:

$246 \times 175 = \mathbf{43050}$

f6	×	v	af	=	?
Calculate Clear					

Figure 42: Braun SIM Calc 4 [5]

Wallace

all code in appendix 2

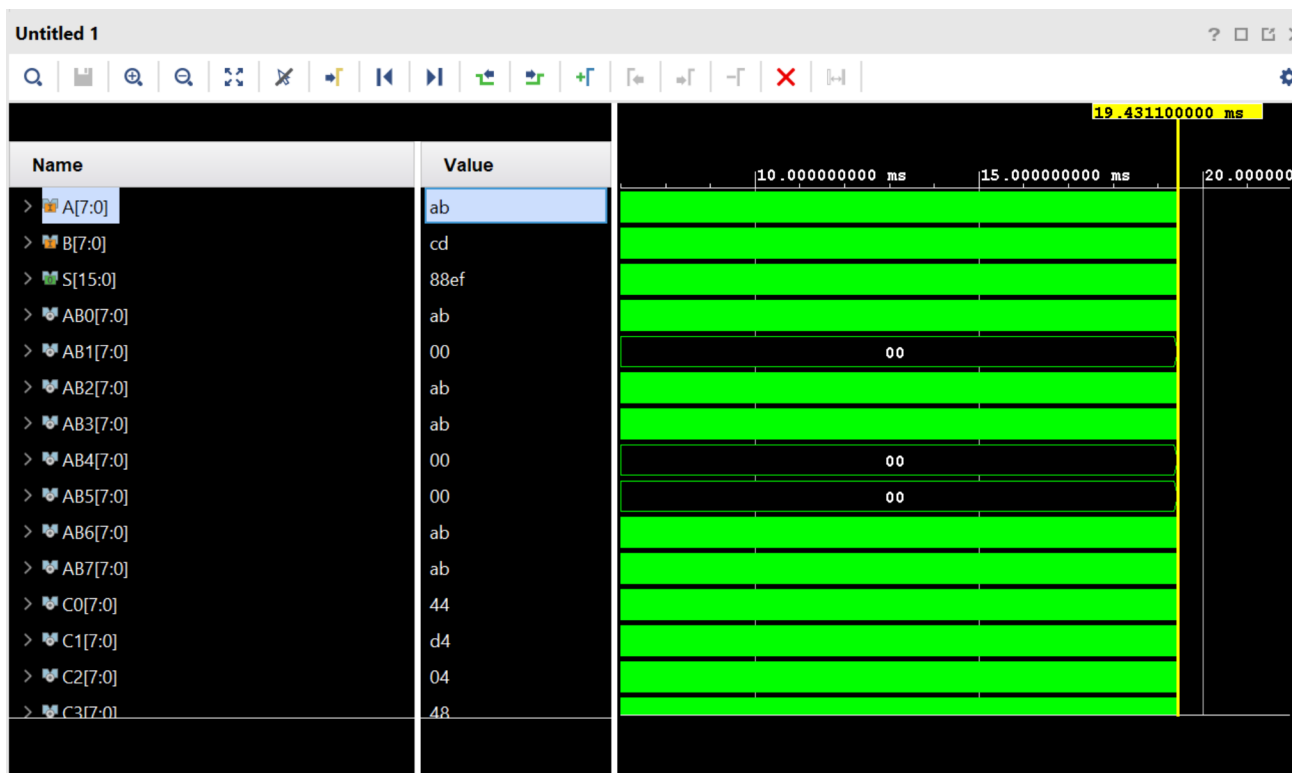


Figure 43: Wallace SIM test 1

the author set the program to clock and input of ab and 00 for A CD and 00 for B this outputs 88ef when converting this to unsigned decimal the author got the same answer.

ab is 171

CD is 205

88ef is 35055

Result save

Hex value:
 $ab \times cd = 88EF$

Decimal value:
 $171 \times 205 = 35055$

ab

x v

cd

= ?

Calculate

Clear

Figure 44: Wallace Calc test 1 [5]

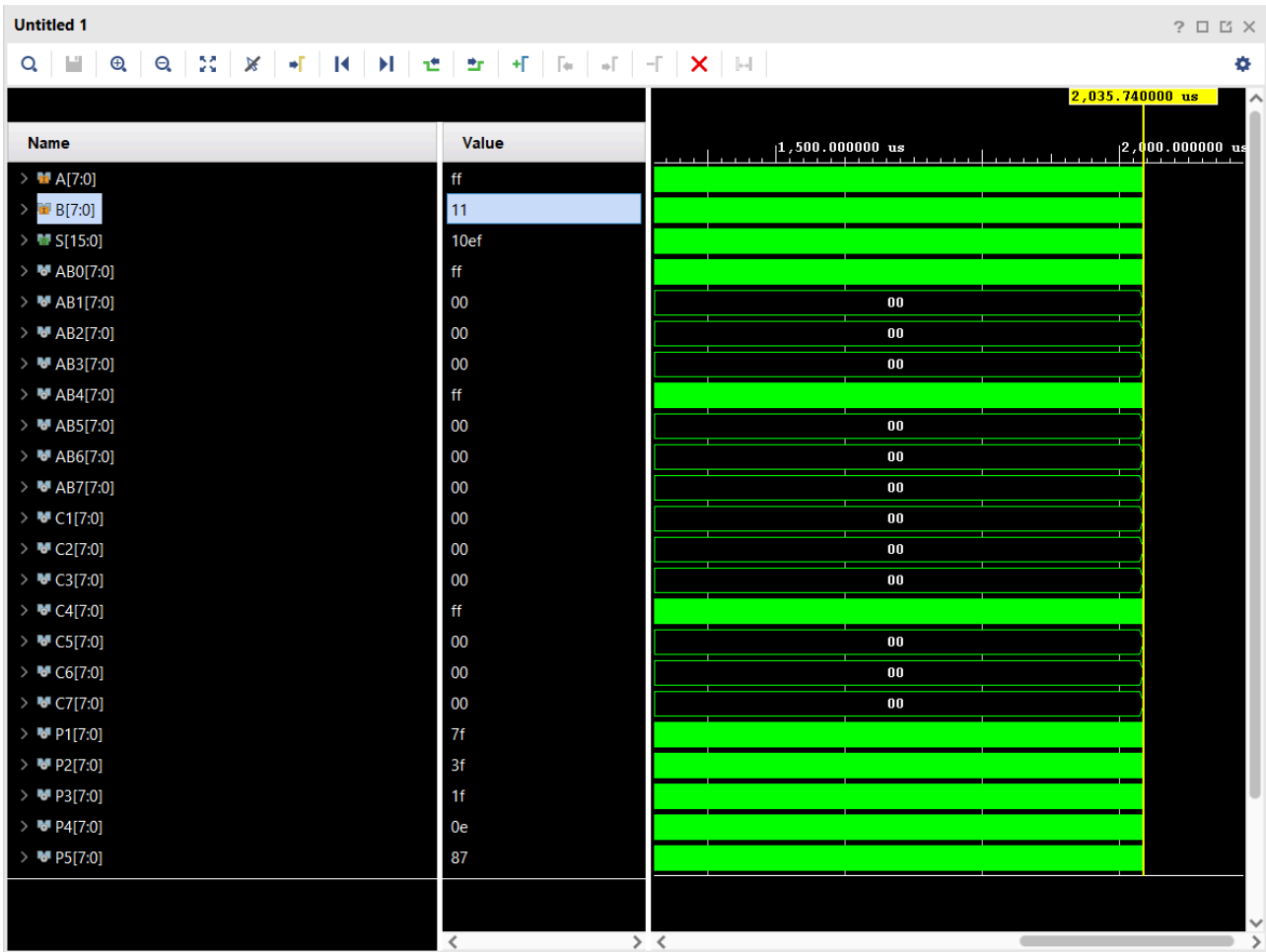


Figure 45: Wallace sim test2

The author set the program to clock and input of ff and 00 for A 11 and 00 for B this outputs 10ef when converting this to unsigned decimal the author got the same answer.

ff is 255
11 is 17
10ef is 4335

Result



Hex value:

ff × 11 = **10EF**

Decimal value:

255 × 17 = **4335**

ff	×	11	= ?
Calculate			Clear

Figure 46: Wallace SIM Calc 2 [5]

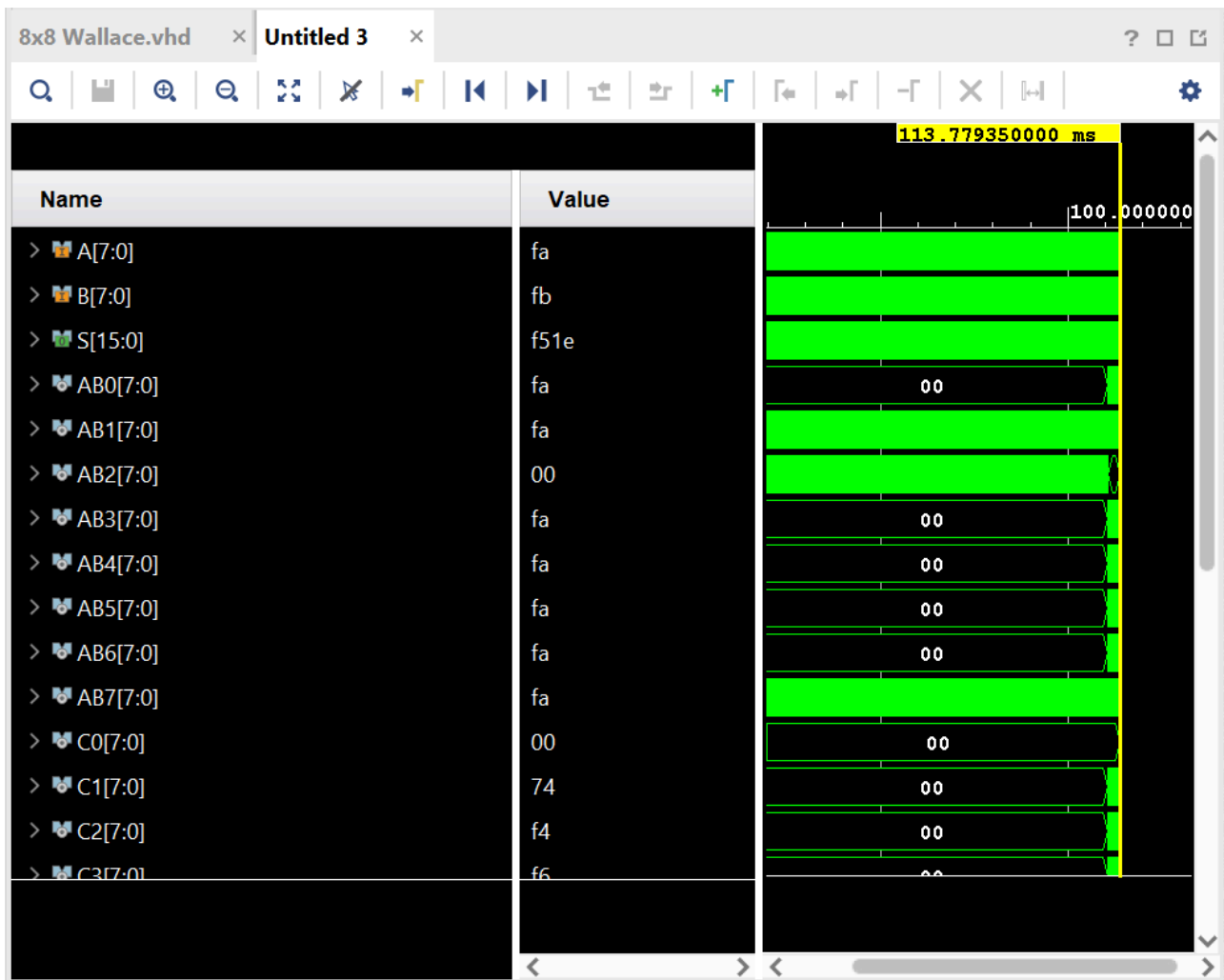


Figure 47: Wallace SIM test 3

fa = 250

fb = 251

f51e = 62750

Result

save

Hex value:
fa × fb = **F51E**

Decimal value:
250 × 251 = **62750**

fa

×

fb

= ?

Calculate

Clear

Figure 48: Wallace SIM Calc 3 [5]

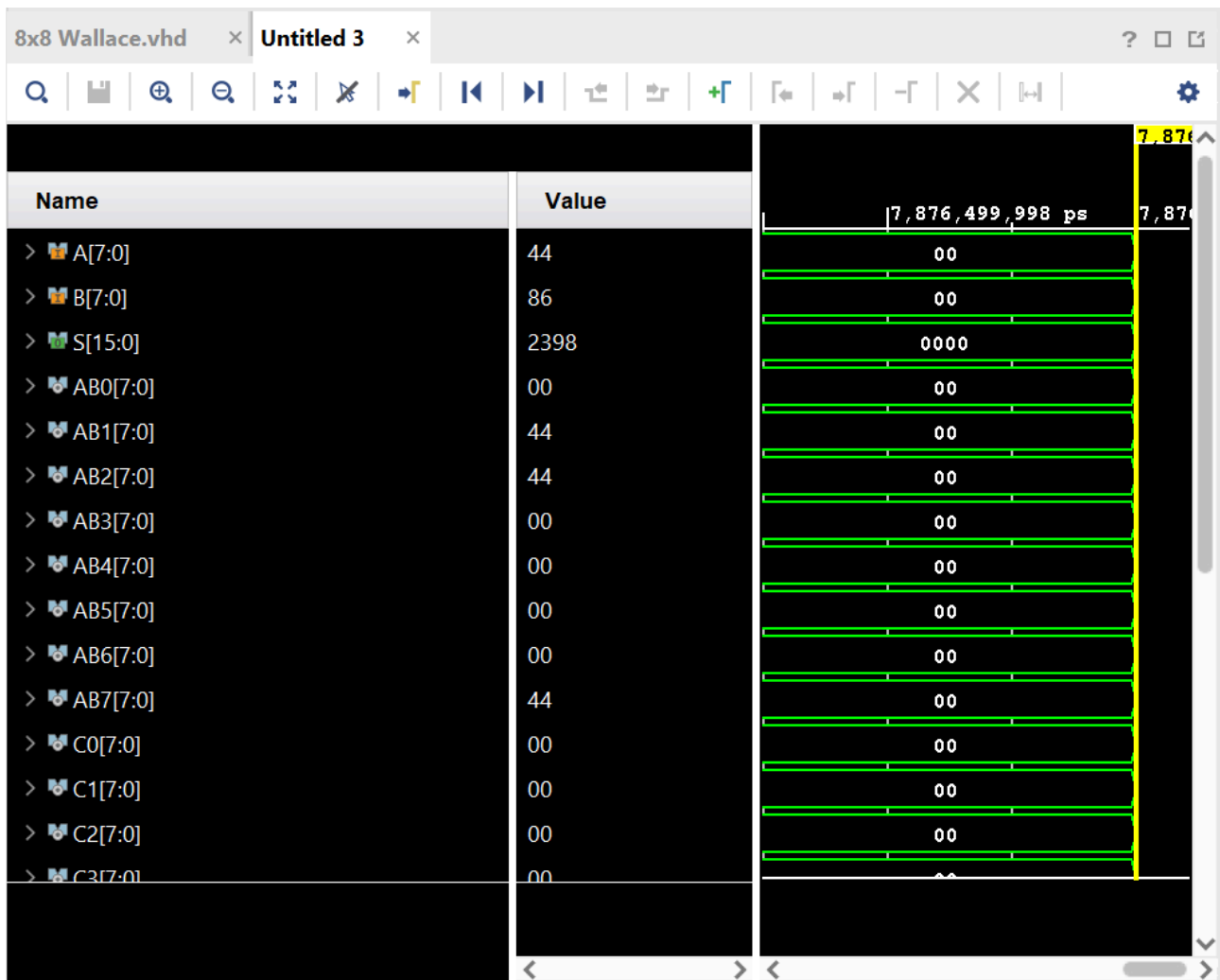


Figure 49: Wallace SIM test 4

44 = 68

86 = 134

2198 = 9112

Result
save

Hex value:
 $44 \times 86 = \mathbf{2398}$

Decimal value:
 $68 \times 134 = \mathbf{9112}$

×

= ?

Calculate

Clear

Figure 50: Wallace SIM Calc 4[5]

Evaluation

The author in this section evaluates the Braun and Wallace multipliers from design to final results of their testing as well as a critical evaluation of their VHDL development in Vivado

Design and implementation

During the design phase the author started by designing the components of their VHDL this makes it easier to develop later and allows the author to have a greater understanding of the adders before creating the block diagram for the Wallace and barn.

The block diagram for Wallace and Braun showcases how the author used their knowledge of the adders to map the outputs and inputs to assist with early development.

This focus on the adder VHDL made the k maps the author made less relevant as they used the truth table and the VHDL code for their adders to develop their block diagram.

Timing

The authors calculated the longest route in their block diagram. They then calculated the time of these longest routes using assumed times for a half and full adder. For Wallace the longest delay was 26ns and for Braun it was 47ns. This is then visible in the timing tests as Braun is shown in both synthesis and implemented designs to be slower by nearly double of Wallace. This matches the longest route being nearly double Wallace's.

power consumption

the author gives a power budget of 15w for both Wallace and Braun they use most of their power in their IO with 0.95 for Wallace used on logic and 1.408 for signals the board uses 11w at maximum this is similar to Braun with it also using around 11w and most of its power in its signals this data is shown in figure 24,30 above.

speed and power efficiency

Due to the speed Wallace processes at the and its lower power usage in the authors implementation it is by far the most power efficient and fastest. although the differences in power are very minimal so most of its efficiency comes from its speed.

LUTs

The Wallace multiplier uses 23 less LUTs that the Braun multiplier making significantly more space efficient than the Braun multiplier. This is shown in figure 25 and figure 29 above for both Braun and Wallace and can be seen on the device image for both above figure 15 figure 16. this would only benefit Wallace as it increases in size and the reduced size is part of the reason it uses less power.

FPGA implementation

When implementing it on FPGA the author decided to use the switches provided and the led's above them to input and show output respectively. This enabled them to input 8 binary numbers to A and B as well as get 16 led outputs on S port. This was simplistic and effective as a route.

Tests

The test method of inputting a hex value to the simulation to test the device made it quick and simple to identify bugs. After these tests demonstrating it on FPGA showed that it was not just the software informing that it was functional and that it worked on physical hardware.

Tools and techniques

The author used Vivado as an industry standard tool and developed their knowledge of VHDL development the authors use of Vivado. shows an understanding of FPGA development and VHDL syntax.

When implementing Braun and Wallace in Vivado it is similar in approach the components are the same; the difference between the two comes in the inputs and outputs of the components. in Braun the components in each bank feed their carry's to the final adder and then down and their sums move down to the next bank. whereas Wallace moves its carry's down as well as its sums enabling it to shorten the route of the logical operations.

7 Conclusion

This report demonstrates a comparative analysis of Braun and Wallace multipliers and critically evaluates their function in both VHDL synthesis via the Vivado software and implementation on the basys3 board.

The findings reveal that Braun multiplier significantly outperform Braun multiplier in terms of power efficiency utilisation efficiency and speed. This makes it a suitable multiplier to use on fpga boards that commonly lack space.

In summary the Wallace multiplier is a preferred choice when selecting between the two multipliers given due to its characteristics of being faster more efficient and smaller. The only time Braun performs better in these cases is with partial computation where the Braun multiplier completes each calculation at the end of each bank compared to the Wallace method that will only be valid at the output.

Bibliography

[1]

P. Mishra, "A study on Wallace tree multiplier," vol. 7, no. 5, Apr. 2018, Accessed: Feb. 28, 2025. [Online]. Available:

https://www.researchgate.net/publication/327209221_A_study_on_Wallace_tree_multiplier

[2]

Magnus Sjölander, "Efficient Reconfigurable Multipliers Based on the Twin-Precision Technique," *Research gate*, Jan. 01, 2006.

https://www.researchgate.net/publication/228867197_Efficient_Reconfigurable_Multipliers_Based_on_the_Twin-Precision_Technique (accessed Feb. 28, 2025).

[3]

Lattice Semiconductor Corporation, "www.latticesemi.com 1 tn1000_09.1 sysIO Usage Guidelines for Lattice Devices," *Digchip.com*, 2025. <https://application-notes.digchip.com/030/30-20552.pdf> (accessed Mar. 21, 2025).

[4]

"Hex Calculator," *Calculator.net*, 2025. <https://www.calculator.net/hex-calculator.html?number1=ab&c2op=x&number2=cd&calctype=op&x=Calculate> (accessed Mar. 21, 2025).

[5]

S. D. Brown and Z. G. Vranesic, *Fundamentals of Digital Logic with VHDL Design*. 2005.

[6]

B. Mealy and F. Tappero, *Free Range VHDL*. 2023. Accessed: Apr. 08, 2025. [Online]. Available: https://faculty-web.msoe.edu/johnsontimoj/EE3921/files3921/Book_FreeRangeVHDL.pdf

[7]

P. A. Freiburger and M. R. Swaine, "Harvard Mark I | computer technology," *Encyclopædia Britannica*. Dec. 18, 2014. Available: <https://www.britannica.com/technology/Harvard-Mark-I>

[8]

"Anton Braun – Computer Timeline," *Computer-timeline.com*, 2025. <http://www.computer-timeline.com/timeline/anton-braun/> (accessed Apr. 10, 2025).

[9]

D. S. ROY, "Wallace and Dedda Multiplier Design - Digital System Design," *Digital System Design*, Jun. 18, 2019. <https://digitalsystemdesign.in/wallace-and-dedda-multiplier-design/> (accessed Apr. 10, 2025).

[10]

M. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann, 2003.

[11]

V. G. and R. Hegadi, "Design and Development of 8-Bits Fast Multiplier for Low Power Applications," *International Journal of Engineering and Technology*, vol. 4, no. 6, pp. 774–780, 2012, doi: <https://doi.org/10.7763/ijet.2012.v4.482>.

Appendix 1

code
Braun

braunmulti8x8.vhd

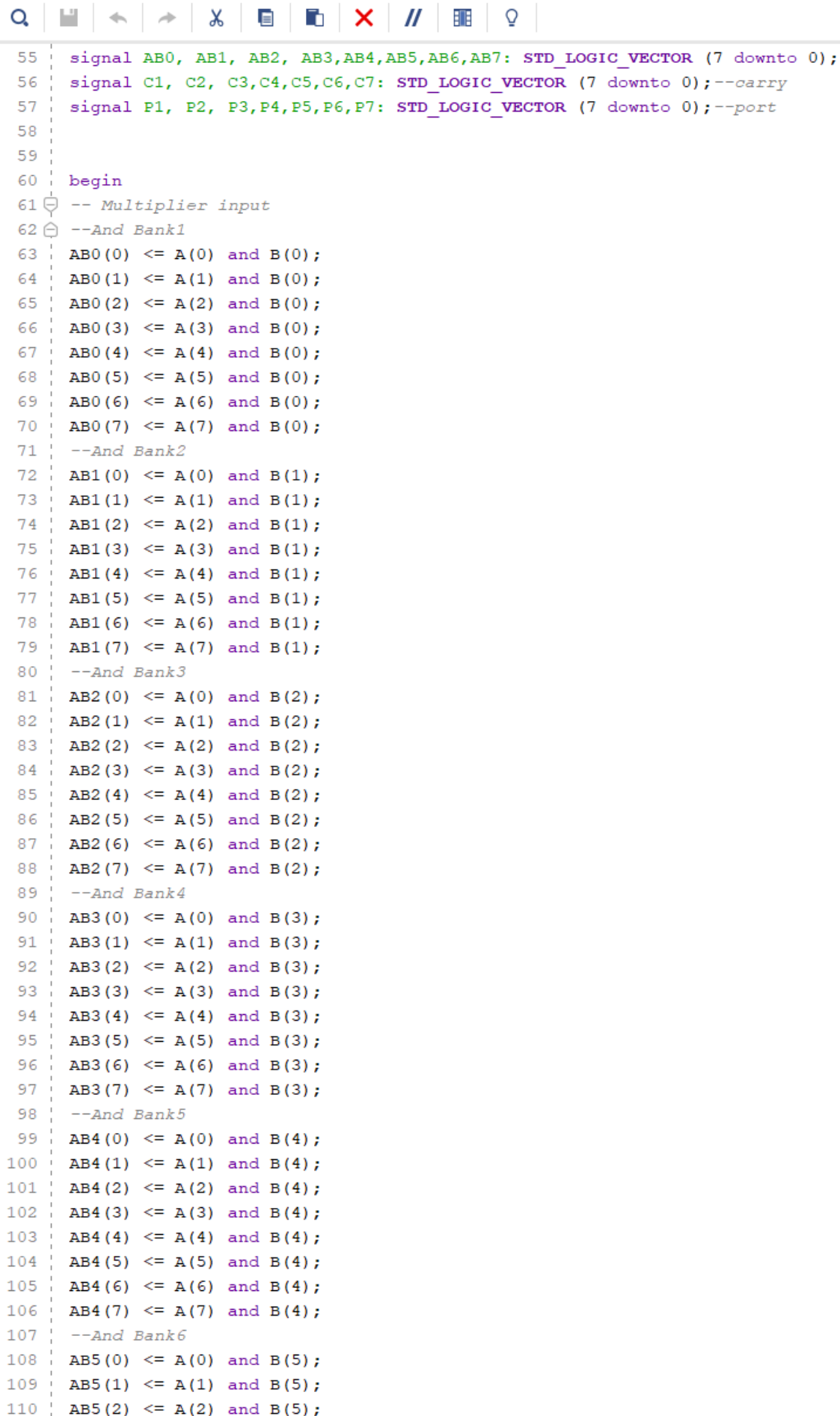
C:/Users/rattb/Electronics2_multipliers/braunmulti8x8.vhd

```
1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 19.02.2025 16:29:20
6  -- Design Name:
7  -- Module Name: braunmulti8x8 - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
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 braunmulti8x8 is
35     port ( A: in STD_LOGIC_VECTOR (7 downto 0);
36           B: in STD_LOGIC_VECTOR (7 downto 0);
37           S: out STD_LOGIC_VECTOR (15 downto 0));
38 end braunmulti8x8;
39 architecture Gate_Level of braunmulti8x8 is
40     -- Full Adder 1bit Component Declaration
41     component full_adder_vhdl_code
42     port ( A,B: in STD_LOGIC;
43           Cin: in STD_LOGIC;
44           Cout: out STD_LOGIC;
45           S: out STD_LOGIC);
46 end component;
47
48     -- Half Adder 1bit Component Declaration
49     component half_adder_vhdl_code
50     port ( A,B: in STD_LOGIC;
51           Cout: out STD_LOGIC;
52           S: out STD_LOGIC);
53 end component;
54
55     -- Intermediate declaration
56     signal AB0, AB1, AB2, AB3, AB4, AB5, AB6, AB7: STD_LOGIC_VECTOR (7 downto 0);
57     signal C1, C2, C3, C4, C5, C6, C7: STD_LOGIC_VECTOR (7 downto 0); --carry
```

Figure 51: Braun code 1

braunmulti8x8.vhd

C:/Users/rattb/Electronics2_multipliers/braunmulti8x8.vhd

The image shows a screenshot of a VHDL code editor. At the top, the file name 'braunmulti8x8.vhd' is displayed in a grey header bar. Below the header, the file path 'C:/Users/rattb/Electronics2_multipliers/braunmulti8x8.vhd' is shown. A toolbar with various icons (search, save, undo, redo, cut, copy, paste, delete, comment, help) is located below the path. The main area contains VHDL code for a multiplier. The code starts with three signal declarations for 8-bit logic vectors: AB0-AB7, C1-C7, and P1-P7. It then enters a 'begin' block. Inside, there are comments for 'Multiplier input', 'And Bank1', 'And Bank2', 'And Bank3', 'And Bank4', and 'And Bank5'. Each bank contains eight assignment statements, each performing a bitwise AND operation between a bit of A and a bit of B, and assigning the result to a corresponding bit of the AB array. For example, in Bank1, AB0(0) is assigned A(0) and B(0). The code ends at line 110 with the assignment for AB5(2).

```
55 signal AB0, AB1, AB2, AB3,AB4,AB5,AB6,AB7: STD_LOGIC_VECTOR (7 downto 0);
56 signal C1, C2, C3,C4,C5,C6,C7: STD_LOGIC_VECTOR (7 downto 0);--carry
57 signal P1, P2, P3,P4,P5,P6,P7: STD_LOGIC_VECTOR (7 downto 0);--port
58
59
60 begin
61 -- Multiplier input
62 --And Bank1
63 AB0(0) <= A(0) and B(0);
64 AB0(1) <= A(1) and B(0);
65 AB0(2) <= A(2) and B(0);
66 AB0(3) <= A(3) and B(0);
67 AB0(4) <= A(4) and B(0);
68 AB0(5) <= A(5) and B(0);
69 AB0(6) <= A(6) and B(0);
70 AB0(7) <= A(7) and B(0);
71 --And Bank2
72 AB1(0) <= A(0) and B(1);
73 AB1(1) <= A(1) and B(1);
74 AB1(2) <= A(2) and B(1);
75 AB1(3) <= A(3) and B(1);
76 AB1(4) <= A(4) and B(1);
77 AB1(5) <= A(5) and B(1);
78 AB1(6) <= A(6) and B(1);
79 AB1(7) <= A(7) and B(1);
80 --And Bank3
81 AB2(0) <= A(0) and B(2);
82 AB2(1) <= A(1) and B(2);
83 AB2(2) <= A(2) and B(2);
84 AB2(3) <= A(3) and B(2);
85 AB2(4) <= A(4) and B(2);
86 AB2(5) <= A(5) and B(2);
87 AB2(6) <= A(6) and B(2);
88 AB2(7) <= A(7) and B(2);
89 --And Bank4
90 AB3(0) <= A(0) and B(3);
91 AB3(1) <= A(1) and B(3);
92 AB3(2) <= A(2) and B(3);
93 AB3(3) <= A(3) and B(3);
94 AB3(4) <= A(4) and B(3);
95 AB3(5) <= A(5) and B(3);
96 AB3(6) <= A(6) and B(3);
97 AB3(7) <= A(7) and B(3);
98 --And Bank5
99 AB4(0) <= A(0) and B(4);
100 AB4(1) <= A(1) and B(4);
101 AB4(2) <= A(2) and B(4);
102 AB4(3) <= A(3) and B(4);
103 AB4(4) <= A(4) and B(4);
104 AB4(5) <= A(5) and B(4);
105 AB4(6) <= A(6) and B(4);
106 AB4(7) <= A(7) and B(4);
107 --And Bank6
108 AB5(0) <= A(0) and B(5);
109 AB5(1) <= A(1) and B(5);
110 AB5(2) <= A(2) and B(5);
```

Figure 52: Braun code 2

braunmulti8x8.vhd

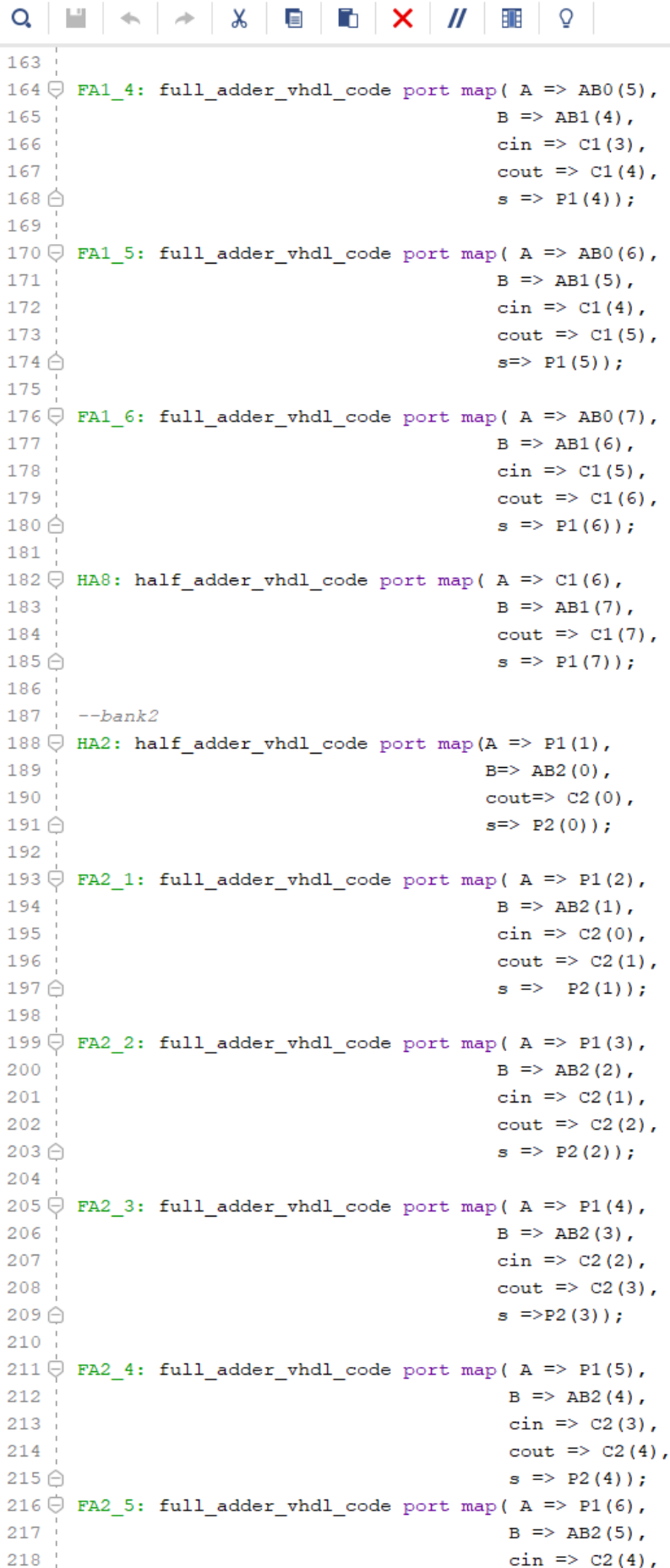
C:/Users/rattb/Electronics2_multipliers/braunmulti8x8.vhd

```
109 AB5(1) <= A(1) and B(5);
110 AB5(2) <= A(2) and B(5);
111 AB5(3) <= A(3) and B(5);
112 AB5(4) <= A(4) and B(5);
113 AB5(5) <= A(5) and B(5);
114 AB5(6) <= A(6) and B(5);
115 AB5(7) <= A(7) and B(5);
116 --And Bank7
117 AB6(0) <= A(0) and B(6);
118 AB6(1) <= A(1) and B(6);
119 AB6(2) <= A(2) and B(6);
120 AB6(3) <= A(3) and B(6);
121 AB6(4) <= A(4) and B(6);
122 AB6(5) <= A(5) and B(6);
123 AB6(6) <= A(6) and B(6);
124 AB6(7) <= A(7) and B(6);
125 -- And Bank8
126 AB7(0) <= A(0) and B(7);
127 AB7(1) <= A(1) and B(7);
128 AB7(2) <= A(2) and B(7);
129 AB7(3) <= A(3) and B(7);
130 AB7(4) <= A(4) and B(7);
131 AB7(5) <= A(5) and B(7);
132 AB7(6) <= A(6) and B(7);
133 AB7(7) <= A(7) and B(7);
134 --half addders
135
136
137 --Bank1
138
139 --fulladders
140 --bank1
141 HA1: half_adder_vhdl_code port map(A =>AB0(1),
142                                     B=> AB1(0),
143                                     cout=> C1(0),
144                                     s=> P1(0));
145
146 FA1_1: full_adder_vhdl_code port map( A => AB0(2),
147                                       B => AB1(1),
148                                       cin => C1(0),
149                                       cout => C1(1),
150                                       s => P1(1));
151
152 FA1_2: full_adder_vhdl_code port map( A => AB0(3),
153                                       B => AB1(2),
154                                       cin => C1(1),
155                                       cout => C1(2),
156                                       s => P1(2));
157
158 FA1_3: full_adder_vhdl_code port map( A => AB0(4),
159                                       B => AB1(3),
160                                       cin => C1(2),
161                                       cout => C1(3),
162                                       s => P1(3));
163
164 FA1_4: full_adder_vhdl_code port map( A => AB0(5),
```

Figure 53: Braun code 3

braunmulti8x8.vhd

C:/Users/rattb/Electronics2_multipliers/braunmulti8x8.vhd



```
163
164 FA1_4: full_adder_vhdl_code port map( A => AB0(5),
165                                         B => AB1(4),
166                                         cin => C1(3),
167                                         cout => C1(4),
168                                         s => P1(4));
169
170 FA1_5: full_adder_vhdl_code port map( A => AB0(6),
171                                         B => AB1(5),
172                                         cin => C1(4),
173                                         cout => C1(5),
174                                         s=> P1(5));
175
176 FA1_6: full_adder_vhdl_code port map( A => AB0(7),
177                                         B => AB1(6),
178                                         cin => C1(5),
179                                         cout => C1(6),
180                                         s => P1(6));
181
182 HA8: half_adder_vhdl_code port map( A => C1(6),
183                                         B => AB1(7),
184                                         cout => C1(7),
185                                         s => P1(7));
186
187 --bank2
188 HA2: half_adder_vhdl_code port map(A => P1(1),
189                                         B=> AB2(0),
190                                         cout=> C2(0),
191                                         s=> P2(0));
192
193 FA2_1: full_adder_vhdl_code port map( A => P1(2),
194                                         B => AB2(1),
195                                         cin => C2(0),
196                                         cout => C2(1),
197                                         s => P2(1));
198
199 FA2_2: full_adder_vhdl_code port map( A => P1(3),
200                                         B => AB2(2),
201                                         cin => C2(1),
202                                         cout => C2(2),
203                                         s => P2(2));
204
205 FA2_3: full_adder_vhdl_code port map( A => P1(4),
206                                         B => AB2(3),
207                                         cin => C2(2),
208                                         cout => C2(3),
209                                         s =>P2(3));
210
211 FA2_4: full_adder_vhdl_code port map( A => P1(5),
212                                         B => AB2(4),
213                                         cin => C2(3),
214                                         cout => C2(4),
215                                         s => P2(4));
216 FA2_5: full_adder_vhdl_code port map( A => P1(6),
217                                         B => AB2(5),
218                                         cin => C2(4),
```

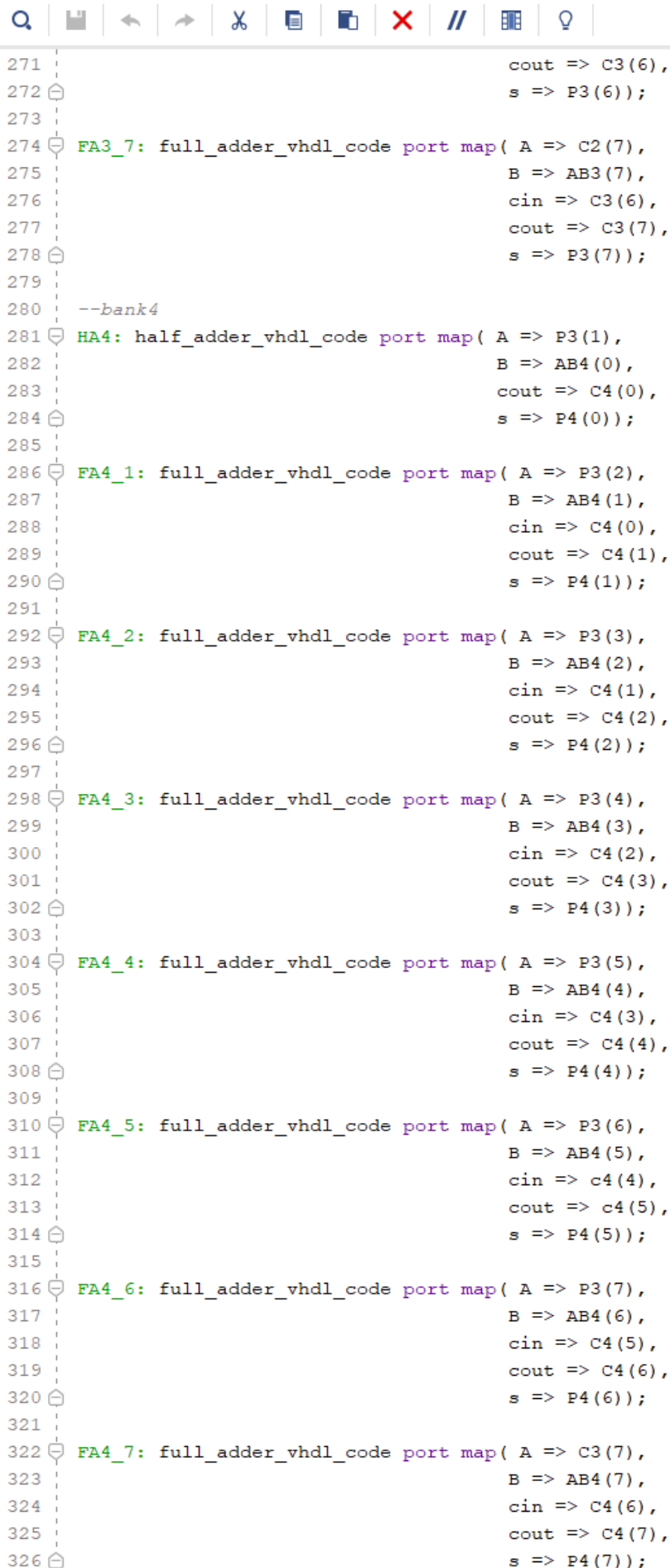
Figure 54: Braun code 4

C:/Users/rattb/Electronics2_multipliers/braunmulti8x8.vhd

Figure 55: Braun code 5

braunmulti8x8.vhd

C:/Users/rattb/Electronics2_multipliers/braunmulti8x8.vhd

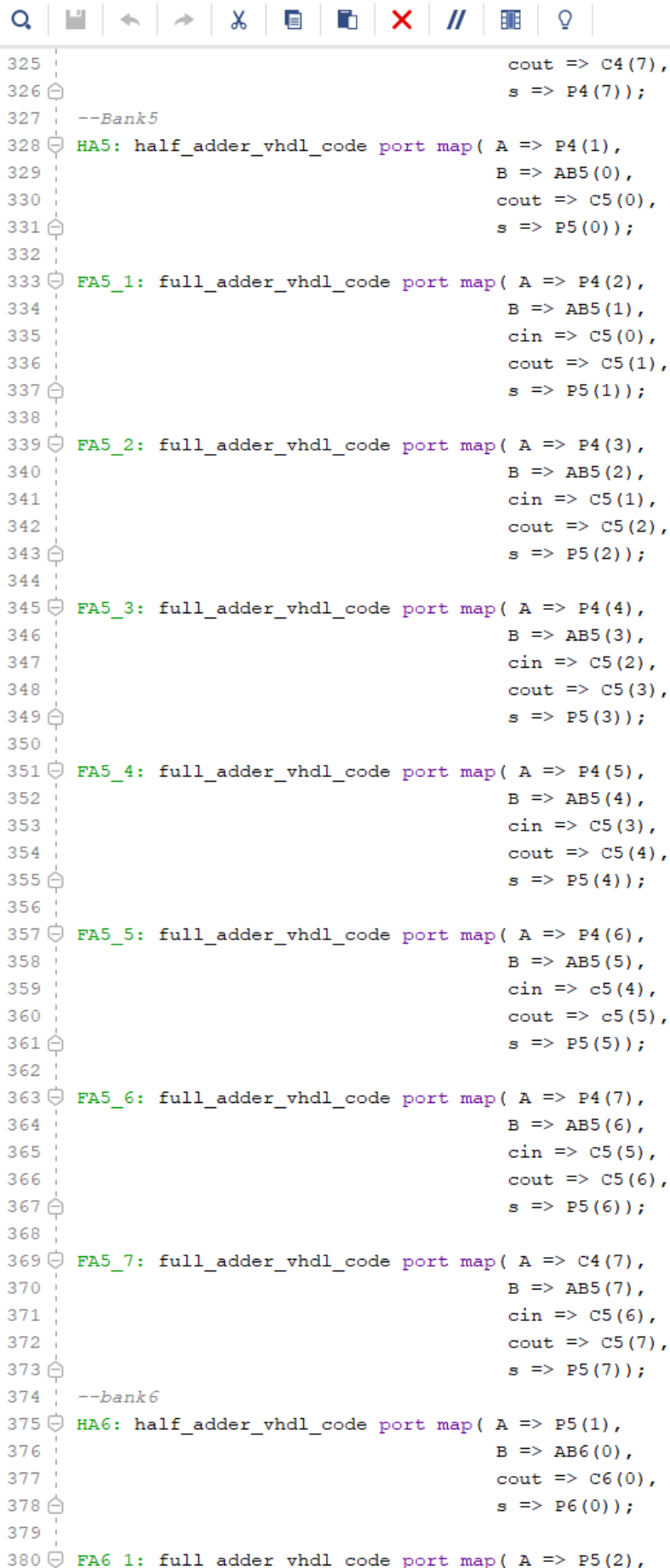


```
271      cout => C3(6),
272      s => P3(6));
273
274  FA3_7: full_adder_vhdl_code port map( A => C2(7),
275      B => AB3(7),
276      cin => C3(6),
277      cout => C3(7),
278      s => P3(7));
279
280  --bank4
281  HA4: half_adder_vhdl_code port map( A => P3(1),
282      B => AB4(0),
283      cout => C4(0),
284      s => P4(0));
285
286  FA4_1: full_adder_vhdl_code port map( A => P3(2),
287      B => AB4(1),
288      cin => C4(0),
289      cout => C4(1),
290      s => P4(1));
291
292  FA4_2: full_adder_vhdl_code port map( A => P3(3),
293      B => AB4(2),
294      cin => C4(1),
295      cout => C4(2),
296      s => P4(2));
297
298  FA4_3: full_adder_vhdl_code port map( A => P3(4),
299      B => AB4(3),
300      cin => C4(2),
301      cout => C4(3),
302      s => P4(3));
303
304  FA4_4: full_adder_vhdl_code port map( A => P3(5),
305      B => AB4(4),
306      cin => C4(3),
307      cout => C4(4),
308      s => P4(4));
309
310  FA4_5: full_adder_vhdl_code port map( A => P3(6),
311      B => AB4(5),
312      cin => C4(4),
313      cout => C4(5),
314      s => P4(5));
315
316  FA4_6: full_adder_vhdl_code port map( A => P3(7),
317      B => AB4(6),
318      cin => C4(5),
319      cout => C4(6),
320      s => P4(6));
321
322  FA4_7: full_adder_vhdl_code port map( A => C3(7),
323      B => AB4(7),
324      cin => C4(6),
325      cout => C4(7),
326      s => P4(7));
```

Figure 56: Braun code 5

braunmulti8x8.vhd

C:/Users/rattb/Electronics2_multipliers/braunmulti8x8.vhd

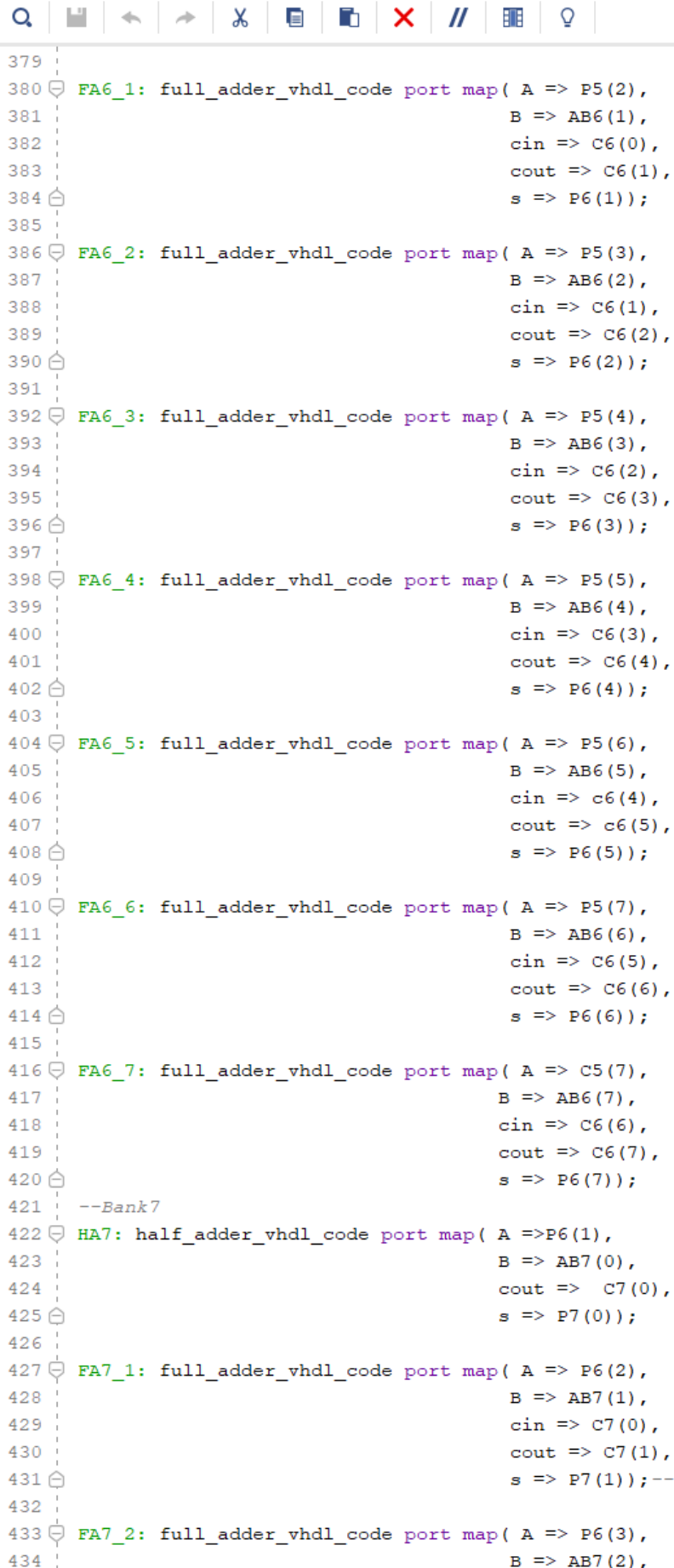


```
325      cout => C4(7),
326      s => P4(7));
327  --Bank5
328  HA5: half_adder_vhdl_code port map( A => P4(1),
329      B => AB5(0),
330      cout => C5(0),
331      s => P5(0));
332
333  FA5_1: full_adder_vhdl_code port map( A => P4(2),
334      B => AB5(1),
335      cin => C5(0),
336      cout => C5(1),
337      s => P5(1));
338
339  FA5_2: full_adder_vhdl_code port map( A => P4(3),
340      B => AB5(2),
341      cin => C5(1),
342      cout => C5(2),
343      s => P5(2));
344
345  FA5_3: full_adder_vhdl_code port map( A => P4(4),
346      B => AB5(3),
347      cin => C5(2),
348      cout => C5(3),
349      s => P5(3));
350
351  FA5_4: full_adder_vhdl_code port map( A => P4(5),
352      B => AB5(4),
353      cin => C5(3),
354      cout => C5(4),
355      s => P5(4));
356
357  FA5_5: full_adder_vhdl_code port map( A => P4(6),
358      B => AB5(5),
359      cin => c5(4),
360      cout => c5(5),
361      s => P5(5));
362
363  FA5_6: full_adder_vhdl_code port map( A => P4(7),
364      B => AB5(6),
365      cin => C5(5),
366      cout => C5(6),
367      s => P5(6));
368
369  FA5_7: full_adder_vhdl_code port map( A => C4(7),
370      B => AB5(7),
371      cin => C5(6),
372      cout => C5(7),
373      s => P5(7));
374  --bank6
375  HA6: half_adder_vhdl_code port map( A => P5(1),
376      B => AB6(0),
377      cout => C6(0),
378      s => P6(0));
379
380  FA6_1: full_adder_vhdl_code port map( A => P5(2),
```

Figure 57: Braun code 6

braunmulti8x8.vhd

C:/Users/rattb/Electronics2_multipliers/braunmulti8x8.vhd



```
379
380 FA6_1: full_adder_vhdl_code port map( A => P5(2),
381                                         B => AB6(1),
382                                         cin => C6(0),
383                                         cout => C6(1),
384                                         s => P6(1));
385
386 FA6_2: full_adder_vhdl_code port map( A => P5(3),
387                                         B => AB6(2),
388                                         cin => C6(1),
389                                         cout => C6(2),
390                                         s => P6(2));
391
392 FA6_3: full_adder_vhdl_code port map( A => P5(4),
393                                         B => AB6(3),
394                                         cin => C6(2),
395                                         cout => C6(3),
396                                         s => P6(3));
397
398 FA6_4: full_adder_vhdl_code port map( A => P5(5),
399                                         B => AB6(4),
400                                         cin => C6(3),
401                                         cout => C6(4),
402                                         s => P6(4));
403
404 FA6_5: full_adder_vhdl_code port map( A => P5(6),
405                                         B => AB6(5),
406                                         cin => c6(4),
407                                         cout => c6(5),
408                                         s => P6(5));
409
410 FA6_6: full_adder_vhdl_code port map( A => P5(7),
411                                         B => AB6(6),
412                                         cin => C6(5),
413                                         cout => C6(6),
414                                         s => P6(6));
415
416 FA6_7: full_adder_vhdl_code port map( A => C5(7),
417                                         B => AB6(7),
418                                         cin => C6(6),
419                                         cout => C6(7),
420                                         s => P6(7));
421 --Bank7
422 HA7: half_adder_vhdl_code port map( A =>P6(1),
423                                         B => AB7(0),
424                                         cout => C7(0),
425                                         s => P7(0));
426
427 FA7_1: full_adder_vhdl_code port map( A => P6(2),
428                                         B => AB7(1),
429                                         cin => C7(0),
430                                         cout => C7(1),
431                                         s => P7(1));--
432
433 FA7_2: full_adder_vhdl_code port map( A => P6(3),
434                                         B => AB7(2),
```

Figure 58: Braun code 7

braunmulti8x8.vhd

C:/Users/rattb/Electronics2_multipliers/braunmulti8x8.vhd

```
433 FA7_2: full_adder_vhdl_code port map( A => P6(3),
434                                         B => AB7(2),
435                                         cin => C7(1),
436                                         cout => C7(2),
437                                         s => P7(2));
438
439 FA7_3: full_adder_vhdl_code port map( A => P6(4),
440                                         B => AB7(3),
441                                         cin => C7(2),
442                                         cout => C7(3),
443                                         s => P7(3));
444
445 FA7_4: full_adder_vhdl_code port map( A => P6(5),
446                                         B => AB7(4),
447                                         cin => C7(3),
448                                         cout => C7(4),
449                                         s => P7(4));
450
451 FA7_5: full_adder_vhdl_code port map( A => P6(6),
452                                         B => AB7(5),
453                                         cin => c7(4),
454                                         cout => c7(5),
455                                         s => P7(5));
456
457 FA7_6: full_adder_vhdl_code port map( A => P6(7),
458                                         B => AB7(6),
459                                         cin => C7(5),
460                                         cout => C7(6),
461                                         s => P7(6));
462
463 FA7_7: full_adder_vhdl_code port map( A => C6(7),
464                                         B => AB7(7),
465                                         cin => C7(6),
466                                         cout => C7(7),
467                                         s => P7(7));
468
469
470
471 -- Mulitplier output
472 S(0) <= AB0(0);
473 S(1) <= P1(0);
474 S(2) <= P2(0);
475 S(3) <= P3(0);
476 S(4) <= P4(0);
477 S(5) <= P5(0);
478 S(6) <= P6(0);
479 S(7) <= P7(0);
480 S(8) <= P7(1);
481 S(9) <= P7(2);
482 S(10) <= P7(3);
483 S(11) <= P7(4);
484 S(12) <= P7(5);
485 S(13) <= P7(6);
486 S(14) <= P7(7);
487 S(15) <= C7(7);
488 end Gate_Level;
```

Figure 59: Braun code 8

Appendix 2

Wallace

```

8x8 Wallace.vhd

C:/Users/rattb/Electronics2_multipliers/Electronics2_multipliers.srcs/sources_1/new/8x8 Wallace.vhd

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 20.02.2025 16:12:49
6  -- Design Name:
7  -- Module Name: Wallace_multiplier8bit - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
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_multiplier8bit is
35     Port ( A : in STD_LOGIC_VECTOR(7 downto 0);
36           B : in STD_LOGIC_VECTOR(7 downto 0);
37           S : out STD_LOGIC_VECTOR(15 downto 0));
38 end Wallace_multiplier8bit;
39
40 architecture Gate_Level of Wallace_multiplier8bit is
41     -- componuts
42     -- Full Adder 1bit Component Decalaration
43     component full_adder_vhdl_code
44     port ( A,B: in STD_LOGIC;
45           Cin: in STD_LOGIC;
46           Cout: out STD_LOGIC;
47           S: out STD_LOGIC);
48 end component;
49
50 -- Half Adder 1bit Component Decalaration
51 component Half_adder_vhdl_code
52 port ( A,B: in STD_LOGIC;
53       Cout: out STD_LOGIC;
54       S: out STD_LOGIC);
55 end component;
56

```

Figure 60: Wallace code 1

```

56
57 signal AB0,AB1,AB2,AB3,AB4,AB5,AB6,AB7 :STD_LOGIC_VECTOR(7 downto 0);
58 signal C0,C1,C2,C3,C4,C5,C6:STD_LOGIC_VECTOR(7 downto 0);
59 signal P0,P1,P2,P3,P4,P5,P6:STD_LOGIC_VECTOR(7 downto 0);
60
61 begin
62 AB0(0) <= A(0) and B(0);
63 AB0(1) <= A(1) and B(0);
64 AB0(2) <= A(2) and B(0);
65 AB0(3) <= A(3) and B(0);
66 AB0(4) <= A(4) and B(0);
67 AB0(5) <= A(5) and B(0);
68 AB0(6) <= A(6) and B(0);
69 AB0(7) <= A(7) and B(0);
70 --bank2
71 AB1(0) <= A(0) and B(1);
72 AB1(1) <= A(1) and B(1);
73 AB1(2) <= A(2) and B(1);
74 AB1(3) <= A(3) and B(1);
75 AB1(4) <= A(4) and B(1);
76 AB1(5) <= A(5) and B(1);
77 AB1(6) <= A(6) and B(1);
78 AB1(7) <= A(7) and B(1);
79 --bank3
80 AB2(0) <= A(0) and B(2);
81 AB2(1) <= A(1) and B(2);
82 AB2(2) <= A(2) and B(2);
83 AB2(3) <= A(3) and B(2);
84 AB2(4) <= A(4) and B(2);
85 AB2(5) <= A(5) and B(2);
86 AB2(6) <= A(6) and B(2);
87 AB2(7) <= A(7) and B(2);
88 --bank4
89 AB3(0) <= A(0) and B(3);
90 AB3(1) <= A(1) and B(3);
91 AB3(2) <= A(2) and B(3);
92 AB3(3) <= A(3) and B(3);
93 AB3(4) <= A(4) and B(3);
94 AB3(5) <= A(5) and B(3);
95 AB3(6) <= A(6) and B(3);
96 AB3(7) <= A(7) and B(3);
97 --bank5
98 AB4(0) <= A(0) and B(4);
99 AB4(1) <= A(1) and B(4);
100 AB4(2) <= A(2) and B(4);
101 AB4(3) <= A(3) and B(4);
102 AB4(4) <= A(4) and B(4);
103 AB4(5) <= A(5) and B(4);
104 AB4(6) <= A(6) and B(4);
105 AB4(7) <= A(7) and B(4);
106 --bank6
107 AB5(0) <= A(0) and B(5);
108 AB5(1) <= A(1) and B(5);

```


8x8 Wallace.vhd

C:/Users/rattb/Electronics2_multipliers/Electronics2_multipliers.srcs/sources_1/new/8x8 W

```

148         B => AB2(1),
149         cin => AB3(0),
150         cout => C0(2),
151     s => P0(2));
152
153 FA1_3: full_adder_vhdl_code port map( A => AB2(2),
154         B => AB3(1),
155         cin => AB4(0),
156         cout => C0(3),
157     s => P0(3));
158
159 FA1_4: full_adder_vhdl_code port map( A => AB3(2),
160         B => AB4(1),
161         cin => AB5(0),
162         cout => C0(4),
163     s => P0(4));
164
165 FA1_5: full_adder_vhdl_code port map( A => AB4(2),
166         B => AB5(1),
167         cin => AB6(0),
168         cout => C0(5),
169     s => P0(5));
170
171 FA1_6: full_adder_vhdl_code port map( A => AB5(2),
172         B => AB6(1),
173         cin => AB7(0),
174         cout => C0(6),
175     s => P0(6));
176
177 HA1_8: half_adder_vhdl_code port map( A => AB6(2),
178         B => AB7(1),
179         cout => C0(7),
180     s => P0(7));
181
182 --bank2
183 HA2: half_adder_vhdl_code port map( A => C0(0),
184         B => P0(1),
185         cout => C1(0),
186     s => P1(0));
187
188 FA2_1: full_adder_vhdl_code port map( A => C0(1),
189         B => P0(2),
190         cin => AB0(3),
191         cout => C1(1),
192     s => P1(1));
193
194 FA2_2: full_adder_vhdl_code port map( A => C0(2),
195         B => P0(3),
196         cin => AB1(3),
197         cout => C1(2),
198     s => P1(2));
199
200 FA2_3: full_adder_vhdl_code port map( A => C0(3),
201         B => P0(4),
202         cin => AB2(3),
203         cout => C1(3),

```

Fig

Figure 63: Wallace code 4
ALEX Glavatt (221/170)

8x8 Wallace.vhd

C:/Users/rattb/Electronics2_multipliers/Electronics2_multipliers.srscs/sources_1/new/8x8 Wallace

```
199
200 FA2_3: full_adder_vhdl_code port map( A => C0(3),
201                                         B => P0(4),
202                                         cin => AB2(3),
203                                         cout => C1(3),
204                                         s => P1(3));
205
206 FA2_4: full_adder_vhdl_code port map( A => C0(4),
207                                         B => P0(5),
208                                         cin => AB3(3),
209                                         cout => C1(4),
210                                         s => P1(4));
211
212 FA2_5: full_adder_vhdl_code port map( A => C0(5),
213                                         B => P0(6),
214                                         cin => AB4(3),
215                                         cout => C1(5),
216                                         s => P1(5));
217
218 FA2_6: full_adder_vhdl_code port map( A => C0(6),
219                                         B => P0(7),
220                                         cin => AB5(3),
221                                         cout => C1(6),
222                                         s => P1(6));
223
224 FA2_7: full_adder_vhdl_code port map( A => C0(7),
225                                         B => AB6(3),
226                                         cin => AB7(2),
227                                         cout => C1(7),
228                                         s => P1(7));
229 --bank3
230 HA3: half_adder_vhdl_code port map( A => C1(0),
231                                         B => P1(1),
232                                         cout => C2(0),
233                                         s => P2(0));
234
235 FA3_1: full_adder_vhdl_code port map( A => C1(1),
236                                         B => P1(2),
237                                         cin => AB0(4),
238                                         cout => C2(1),
239                                         s => P2(1));
240
241 FA3_2: full_adder_vhdl_code port map( A => C1(2),
242                                         B => P1(3),
243                                         cin => AB1(4),
244                                         cout => C2(2),
245                                         s => P2(2));
246
247 FA3_3: full_adder_vhdl_code port map( A => C1(3),
248                                         B => P1(4),
249                                         cin => AB2(4),
250                                         cout => C2(3),
```

8x8 Wallace.vhd

C:/Users/rattb/Electronics2_multipliers/Electronics2_multipliers.srscs/sour

```
253 FA3_4: full_adder_vhdl_code port map( A => C1(4
254                                     B => P1(5),
255                                     cin => AB3(
256                                     cout => C2(
257                                     s => P2(4))
258
259 FA3_5: full_adder_vhdl_code port map( A => C1(5
260                                     B => P1(6),
261                                     cin => AB4(
262                                     cout => C2(
263                                     s => P2(5))
264
265 FA3_6: full_adder_vhdl_code port map( A => C1(6
266                                     B => p1(7),
267                                     cin => AB5(
268                                     cout => C2(
269                                     s => P2(6))
270
271 FA3_7: full_adder_vhdl_code port map( A => C1(7
272                                     B => AB6(4)
273                                     cin => AB7(
274                                     cout => C2(
275                                     s => P2(7))
276 --bank4
277 HA4: half_adder_vhdl_code port map( A => C2(0),
278                                     B => P2(1),
279                                     cout => C3(
280                                     s => p3(0))
281
282 FA4_1: full_adder_vhdl_code port map( A => C2(1
283                                     B => P2(2),
284                                     cin => AB0(
285                                     cout => C3(
286                                     s => P3(1))
287
288 FA4_2: full_adder_vhdl_code port map( A => C2(2
289                                     B => P2(3),
290                                     cin => AB1(
291                                     cout => C3(
292                                     s => P3(2))
293
294 FA4_3: full_adder_vhdl_code port map( A => C2(3
295                                     B => P2(4),
296                                     cin => AB2(
297                                     cout => C3(
298                                     s => P3(3))
299
300 FA4_4: full_adder_vhdl_code port map( A => C2(4
301                                     B => P2(5),
302                                     cin => AB3(
```

8x8 Wallace.vhd

C:/Users/rattb/Electronics2_multipliers/Electronics2_multipliers.srscs/sources_1/new/

```
307         B => P2(6),
308         cin => AB4(5),
309         cout => C3(5),
310         s => P3(5));
311
312 FA4_6: full_adder_vhdl_code port map( A => C2(6),
313         B => p2(7),
314         cin => AB6(5),
315         cout => C3(6),
316         s => P3(6));
317
318 FA4_7: full_adder_vhdl_code port map( A => C2(7),
319         B => AB6(5),
320         cin => AB7(4),
321         cout => C3(7),
322         s => P3(7));
323 --bank5
324 HA5: half_adder_vhdl_code port map( A => C3(0),
325         B => P3(1),
326         cout => C4(0),
327         s => p4(0));
328
329 FA5_1: full_adder_vhdl_code port map( A => C3(1),
330         B => P3(2),
331         cin => AB0(6),
332         cout => C4(1),
333         s => P4(1));
334
335 FA5_2: full_adder_vhdl_code port map( A => C3(2),
336         B => P3(3),
337         cin => AB1(6),
338         cout => C4(2),
339         s => P4(2));
340
341 FA5_3: full_adder_vhdl_code port map( A => C3(3),
342         B => P3(4),
343         cin => AB2(6),
344         cout => C4(3),
345         s => P4(3));
346
347 FA5_4: full_adder_vhdl_code port map( A => C3(4),
348         B => P3(5),
349         cin => AB3(6),
350         cout => C4(4),
351         s => P4(4));
352
353 FA5_5: full_adder_vhdl_code port map( A => C3(5),
354         B => P3(6),
355         cin => AB4(6),
356         cout => C4(5),
357         s => P4(5));
358
359 FA5_6: full_adder_vhdl_code port map( A => C3(6),
360         B => P3(7),
```

8x8 Wallace.vhd

C:/Users/rattb/Electronics2_multipliers/Electronics2_multipliers.srscs/sources_1/ne

```
361      cin => AB5(6),
362      cout => C4(6),
363      s => P4(6));
364
365  FA5_7: full_adder_vhdl_code port map( A => C3(7),
366      B => AB6(6),
367      cin => AB7(5),
368      cout => C4(7),
369      s => P4(7));
370  --bank6
371  HA6: half_adder_vhdl_code port map( A => C4(0),
372      B => P4(1),
373      cout => C5(0),
374      s => p5(0));
375
376  FA6_1: full_adder_vhdl_code port map( A => C4(1),
377      B => P4(2),
378      cin => AB0(7),
379      cout => C5(1),
380      s => P5(1));
381
382  FA6_2: full_adder_vhdl_code port map( A => C4(2),
383      B => P4(3),
384      cin => AB1(7),
385      cout => C5(2),
386      s => P5(2));
387
388  FA6_3: full_adder_vhdl_code port map( A => C4(3),
389      B => P4(4),
390      cin => AB2(4),
391      cout => c5(3),
392      s => P5(3));
393
394  FA6_4: full_adder_vhdl_code port map( A => c4(4),
395      B => P4(5),
396      cin => AB3(7),
397      cout => C5(4),
398      s => P5(4));
399
400  FA6_5: full_adder_vhdl_code port map( A => C4(5),
401      B => P4(6),
402      cin => AB4(7),
403      cout => C5(5),
404      s => P5(5));
405
406  FA6_6: full_adder_vhdl_code port map( A => C4(6),
407      B => P4(7),
408      cin => AB5(7),
409      cout => C5(6),
410      s => P5(6));
411
```

8x8 Wallace.vhd

C:/Users/rattb/Electronics2_multipliers/Electronics2_multipliers.srscs/sour

```
416      s => P5(7))
417      --bank7
418  HA7: half_adder_vhdl_code port map( A => C5(0),
419      B => P5(1),
420      cout => c6(
421      s => p6(0))
422
423  FA7_1: full_adder_vhdl_code port map( A => C5(1
424      B => P5(2),
425      cin => c6(0
426      cout => c6(
427      s => p6(1))
428
429  FA7_2: full_adder_vhdl_code port map( A => C5(2
430      B => P5(3),
431      cin => c6(1
432      cout => C6(
433      s => p6(2))
434
435  FA7_3: full_adder_vhdl_code port map( A => C5(3
436      B => P5(4),
437      cin => c6(2
438      cout => C6(
439      s => p6(3))
440
441  FA7_4: full_adder_vhdl_code port map( A => C5(4
442      B => P5(5),
443      cin => c6(3
444      cout => C6(
445      s => p6(4))
446
447  FA7_5: full_adder_vhdl_code port map( A => C5(5
448      B => P5(6),
449      cin => c6(4
450      cout => c6(
451      s => p6(5))
452
453  FA7_6: full_adder_vhdl_code port map( A => C5(6
454      B => P5(7),
455      cin => c6(5
456      cout => c6(
457      s => p6(6))
458
459  FA7_7: full_adder_vhdl_code port map( A => C5(7
```

```

464
465  --out
466  S(0) <= AB0(0);
467  S(1) <= p0(0);
468  S(2) <= p1(0);
469  S(3) <= p2(0);
470  S(4) <= p3(0);
471  S(5) <= p4(0);
472  S(6) <= p5(0);
473  S(7) <= p6(0);
474  S(8) <= p6(1);
475  S(9) <= p6(2);
476  S(10) <= p6(3);
477  S(11) <= p6(4);
478  S(12) <= p6(5);
479  S(13) <= p6(6);
480  S(14) <= p6(7);
481  S(15) <= C6(7);
482  end Gate_level;
483

```

Figure 69: Wallace code 10

Appendix 3

the section below changes slightly between Braun and Wallace implementation and synthesis with brauns max delay on line 10 changing to 47.00

Constraints8x8.xdc

C:/Users/rattb/Electronics2_multipliers/Electronics2_multipliers.srscs/constrs_1/new/Constraints8x8.xdc

```
1  # Create a virtual clock (needed for timing analysis)
2
3  #create_clock -period 20.000 -name virtual_clk -waveform {0.000 10.000}
4
5  # Set delays based on analytical calculations
6
7
8
9  # Set max delay for the critical path (longest delay)
10 set_max_delay -from [get_ports A*] -to [get_ports S*] 26.000
11 set_min_delay -from [get_ports A*] -to [get_ports S*] 0.000
12
13
14
15
16
17 set_property PACKAGE_PIN A1 [get_ports {A[7]}]
18 set_property PACKAGE_PIN A3 [get_ports {A[6]}]
19 set_property PACKAGE_PIN A4 [get_ports {A[5]}]
20 set_property PACKAGE_PIN A5 [get_ports {A[4]}]
21 set_property PACKAGE_PIN A6 [get_ports {A[3]}]
22 set_property PACKAGE_PIN A8 [get_ports {A[2]}]
23 set_property PACKAGE_PIN A9 [get_ports {A[1]}]
24 set_property PACKAGE_PIN A10 [get_ports {A[0]}]
25 set_property PACKAGE_PIN A11 [get_ports {B[7]}]
26 set_property PACKAGE_PIN A13 [get_ports {B[6]}]
27 set_property PACKAGE_PIN A14 [get_ports {B[5]}]
28 set_property PACKAGE_PIN A15 [get_ports {B[4]}]
29 set_property PACKAGE_PIN A16 [get_ports {B[3]}]
30 set_property PACKAGE_PIN A18 [get_ports {B[2]}]
31 set_property PACKAGE_PIN B1 [get_ports {B[1]}]
32 set_property PACKAGE_PIN B2 [get_ports {B[0]}]
33 set_property PACKAGE_PIN B3 [get_ports {S[15]}]
34 set_property PACKAGE_PIN B4 [get_ports {S[14]}]
35 set_property PACKAGE_PIN B6 [get_ports {S[13]}]
36 set_property PACKAGE_PIN B7 [get_ports {S[12]}]
37 set_property PACKAGE_PIN B8 [get_ports {S[11]}]
38 set_property PACKAGE_PIN B9 [get_ports {S[10]}]
39 set_property PACKAGE_PIN B11 [get_ports {S[9]}]
40 set_property PACKAGE_PIN B12 [get_ports {S[8]}]
41 set_property PACKAGE_PIN B13 [get_ports {S[7]}]
42 set_property PACKAGE_PIN B14 [get_ports {S[6]}]
43 set_property PACKAGE_PIN B16 [get_ports {S[5]}]
44 set_property PACKAGE_PIN B17 [get_ports {S[4]}]
45 set_property PACKAGE_PIN B18 [get_ports {S[3]}]
46 set_property PACKAGE_PIN C1 [get_ports {S[2]}]
47 set_property PACKAGE_PIN C2 [get_ports {S[1]}]
48 set_property PACKAGE_PIN C4 [get_ports {S[0]}]
49 set_property IOSTANDARD LVCMOS15 [get_ports {S[6]}]
50 set_property IOSTANDARD LVCMOS15 [get_ports {S[5]}]
51 set_property IOSTANDARD LVCMOS15 [get_ports {S[4]}]
52 set_property IOSTANDARD LVCMOS15 [get_ports {S[3]}]
53 set_property IOSTANDARD LVCMOS15 [get_ports {S[2]}]
54 set_property IOSTANDARD LVCMOS15 [get_ports {S[1]}]
55 set_property IOSTANDARD LVCMOS15 [get_ports {S[0]}]
```

```

56 set_property IOSTANDARD LVCMOS15 [get_ports {S[7]}]
57 set_property IOSTANDARD LVCMOS15 [get_ports {S[8]}]
58 set_property IOSTANDARD LVCMOS15 [get_ports {S[9]}]
59 set_property IOSTANDARD LVCMOS15 [get_ports {S[10]}]
60 set_property IOSTANDARD LVCMOS15 [get_ports {S[12]}]
61 set_property IOSTANDARD LVCMOS15 [get_ports {S[11]}]
62 set_property IOSTANDARD LVCMOS15 [get_ports {S[13]}]
63 set_property IOSTANDARD LVCMOS15 [get_ports {S[14]}]
64 set_property IOSTANDARD LVCMOS15 [get_ports {S[15]}]
65 set_property IOSTANDARD LVCMOS15 [get_ports {A[7]}]
66 set_property IOSTANDARD LVCMOS15 [get_ports {A[6]}]
67 set_property IOSTANDARD LVCMOS15 [get_ports {A[5]}]
68 set_property IOSTANDARD LVCMOS15 [get_ports {A[4]}]
69 set_property IOSTANDARD LVCMOS15 [get_ports {A[3]}]
70 set_property IOSTANDARD LVCMOS15 [get_ports {A[2]}]
71 set_property IOSTANDARD LVCMOS15 [get_ports {A[1]}]
72 set_property IOSTANDARD LVCMOS15 [get_ports {A[0]}]
73 set_property IOSTANDARD LVCMOS15 [get_ports {B[7]}]
74 set_property IOSTANDARD LVCMOS15 [get_ports {B[6]}]
75 set_property IOSTANDARD LVCMOS15 [get_ports {B[5]}]
76 set_property IOSTANDARD LVCMOS15 [get_ports {B[4]}]
77 set_property IOSTANDARD LVCMOS15 [get_ports {B[3]}]
78 set_property IOSTANDARD LVCMOS15 [get_ports {B[2]}]
79 set_property IOSTANDARD LVCMOS15 [get_ports {B[1]}]
80 set_property IOSTANDARD LVCMOS15 [get_ports {B[0]}]
81
82 set_property CONFIG_VOLTAGE 1.5 [current_design]
83 set_property CFGBVS GND [current_design]
84
85
86
87 set_operating_conditions -design_power_budget 15.0
88

```

Figure 71: constants code 2