

Electronics 2 Assignment - Ruben Bhatia - 2104693

by Ruben Bhatia

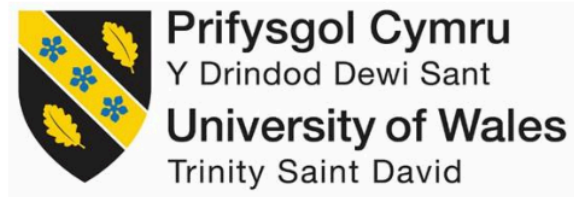
Submission date: 17-Apr-2025 11:25AM (UTC+0100)

Submission ID: 255934481

File name: 33053_Ruben_Bhatia_Electronics_2_Assignment_-_Ruben_Bhatia_-_2104693_654553_1544467892.docx (3.76M)

Word count: 3088

Character count: 16636



Critical Evaluation of Braun's Multiplier and
Wallace Tree Multiplier using VHDL
by
Ruben Bhatia

Assignment submitted as part of the requirements
for the award of
*BEng (Hons) Electrical and Electronics
Engineering*

April 2025

DECLARATION

I, Ruben Bhatia, declare that I am the sole author of this assignment; that all references cited have been consulted; that I have conducted all work of which this is a record, and that the finished work lies within the prescribed word limits.

This work has not previously been accepted as part of any other degree submission.

Date : 13th April 2025

Introduction	3
Background Research of Multiplier Techniques	4
Multipliers & Assumed Advantages	4
Applications of Multipliers	5
Flowchart of Operation.....	5
Methodology	6
Multiplier Diagrams & Algorithms.....	7
Brauns	7
Wallace.....	7
VHDL Code.....	8
Brauns	8
Wallace.....	20
Timings & Schematics	27
Brauns	27
Wallace.....	28
FPGA Design & Pin Allocation.....	31
Brauns	31
Wallace.....	32
Simulation Results	35
Brauns	35
Wallace.....	44
Analysis of Results.....	50
Testing Details & Recommendations for Future Improvements	51
Conclusion	51
References.....	51

Introduction

The aim of this report is to develop, compare and analyse the performance of 8x8 Braun's and Wallace Tree multipliers using VHDL. Both 8x8 multipliers must be designed to allow for the multiplication of two unsigned numbers, where each bit of inputs A and B are multiplied to get a 16 bit result.

To achieve this, diagrams must first be made using appropriate software and research into multipliers. Using these diagrams and the collated research, the multipliers can then be developed using Vivado. Once they are coded and multiply correctly within the software, they can then be evaluated based on their performance by adding timings, looking at their power usage and their operational speed as a whole. From this, they can be compared against each other. These comparisons will dictate which multiplier structure is optimal and should provide justification for why the answer is so.

Background Research of Multiplier Techniques

Multipliers & Assumed Advantages

Multipliers are used in a variety of ways. They aim to perform multiplication between two separate integers. Their structure and usage within FPGAs (Field Programmable Gate Arrays) and thus DSPs (digital signal processing) makes their performance vary. This is dependant on the type of multiplier used [1]. FPGAs use these DSP blocks to incorporate multipliers within them as they are efficient in speed and power usage, meaning that most FPGAs have a dedicated section for DSP block incorporation [2]. This difference in performance of different multiplier types used within FPGAs is impacted by the multipliers power consumption, speed and area [3].

The process of multiplication within an FPGA is done using a 'shift-add' operation. This, in essence, is where the multipliers binary digits are taken one by one from right to left and multiplied by the multiplicand (also in binary) with each single digit of the multiplier. The product of this is then placed to the left of earlier results. This is also known as a logical right shift [4]. This is then repeated for each subsequent digit of the multiplier. The product will also be in binary form [5]. This method of multiplication can only be achieved using unsigned integers.

There are various types of multipliers which operate using different speeds and structures to one another. Wallace Tree multipliers combine their partial products inputted via AND gates into carry and sum outputs produced from their full or half adders. The carry and sum are also collectively known as 'partial products'. These adders make up stages of the tree where one row of adders is a single stage. The outputs are then combined again and are used as inputs into further adders which will produce more carry/sum outputs or a product (a carry or sum bit which doesn't lead to further adders). Wallace Trees also have as many stages as is required to reduce the number of outputs to a single carry or sum [6].

Another type of multiplier is a Brauns multiplier, which is an implementation of a parallel array multiplier. This uses a 'Carry-Save' adder, which is just multiple full adders connected to allow for the binary input of AND gates as well carry/sum inputs produced by other full/half adders to produce further carries/sums/products [7]. Although these are also used in Wallace Tree multipliers, Braun multipliers utilise what is known as a 'Ripple-Carry' adder. Similarly to Carry-Save adders, Ripple-Carry adders are multiple full adders used in conjunction with each other but instead are chained together using the carry input and output ports of the full adders [8]. Where the Wallace Tree multiplier uses these Ripple-Carry adders only on the bottom stage of the tree, the Brauns multiplier uses these throughout its structure. This impacts the speed of the Brauns multipliers as the product of a multiplier its

multiplicand are reliant on the efficiency of operation of each full adder used. As a Wallace multiplier only uses Ripple-Carry adders on the bottom stage of operation, less time is spent waiting on prior full adders to conduct their function, making their speed faster than that of a Brauns multiplier.

Another type of multiplier is a Booth multiplier. It multiplies signed numbers by adding, subtracting or leaving two separate bits as they are (depending on the input bits). The product of this is then dealt with by having its most significant bit copied and its least significant bit lost. This is also known as an arithmetic right shift [4]. It knows whether to add, subtract or leave the bits alone using 'two's complement' and an accumulator/register [9]. Two's complement is a method of which computers can detect and use negative (signed) integers within their operation, where the number is written in binary, then inverted. One is then added to the result of this [10]. The accumulator is used to track whether the previous product of the operation is a 1 or 0 and add, subtract or leave the multiplicand alone. It is important to note that the accumulator is always 0 at the beginning of a Booth multipliers operation [9]. Booths multipliers use adders and subtractors to conduct their operation meaning their structure is larger than that of a Brauns or Wallace multiplier, making them slower. However, it is more efficient in multiplying signed numbers compared to other multipliers.

The final type of multiplier covered for research in this report is the Dadda multiplier. This is a modified version of the Wallace Tree multiplier. A Dadda multiplier requires less hardware and thus conducts operations quicker than that of the Wallace Tree multiplier. It does this by using specific placements of half and full adders within the maximum critical path (the longest path an operation within the multiplier will take). The structure of a Dadda multiplier is split up into three sections. The first section is where partial products are generated using adders. The second section is the partial product summation tree. The third section is the 'final adder' [11]. The first section is comprised of AND gates which produce the sum. These partial products are inputted into full and half adders (connected in parallel) which make up the second section (the partial product summation tree), where three input bits are inputted into a full adder and two input bits are inputted into a half adder [12]. This is where the Dadda tree becomes more optimised for multiplication in comparison to a Wallace Tree multiplier. Once there are two input bits left, these are inputted into the third section of the Dadda multiplier, the final adder. The sum is of this adder is the final product [11].

Applications of Multipliers

Applications of multipliers vary. As multipliers are used in DSPs and thus FPGAs, they can be used in computers ranging from desktops to phones, communication systems and renewable energy automation systems. They can also be used within vehicles, an example of this being FPGAs being used within cruise control systems of a car. As FPGAs allow for the acceleration of data processing [13], multipliers may also be used within networks and the cloud.

Flowchart of Operation

The operational flowchart of an unsigned binary multiplier can be seen below:

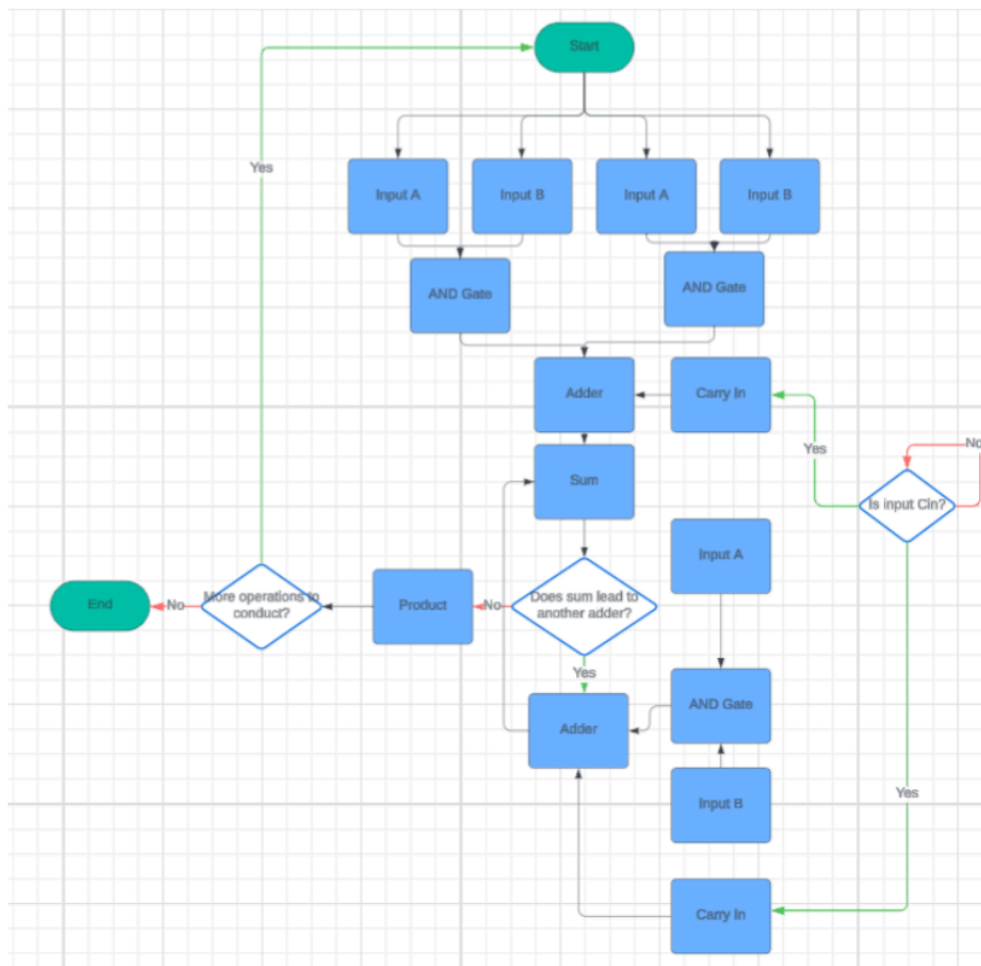


Figure 1

Methodology

The first step in designing both the 8x8 Brauns and Wallace multipliers was to create a diagram of logic flow of each. This was made by conducting research into the separate multipliers and their architecture and implementing it. Once the diagrams were made, code was then written using Vivado, a software which allows for the creation of FPGAs using VHDL. Once the code was written for both separate multipliers, a behavioural simulation was ran within the software. Testing of the multipliers operation then occurred to ascertain whether they were coded and function correctly. This was done multiple times for each multiplier to get an accurate representation of their operation. If multiplications conducted did not provide the correct product, bug fixing would then occur. Following this was the running of the simulated FPGA. Input pins were then selected. Afterwards, an implementation of the FPGA was ran. The timing constrains were then calculated respective to the Brauns and

Wallace Tree multiplier and were added to their individual code. The timing constraint periods (or clock cycles) were calculated using the critical path of each multiplier. This calculated to 41ns for the Brauns multiplier (20 adders + 1 AND gate) and 29ns for the Wallace tree multiplier (14 adders + 1 AND gate). Finally, a timing summary and power summary were generated from the implemented design of both multipliers.

Multiplier Diagrams & Algorithms

Brauns

Here is the Brauns multiplier design:

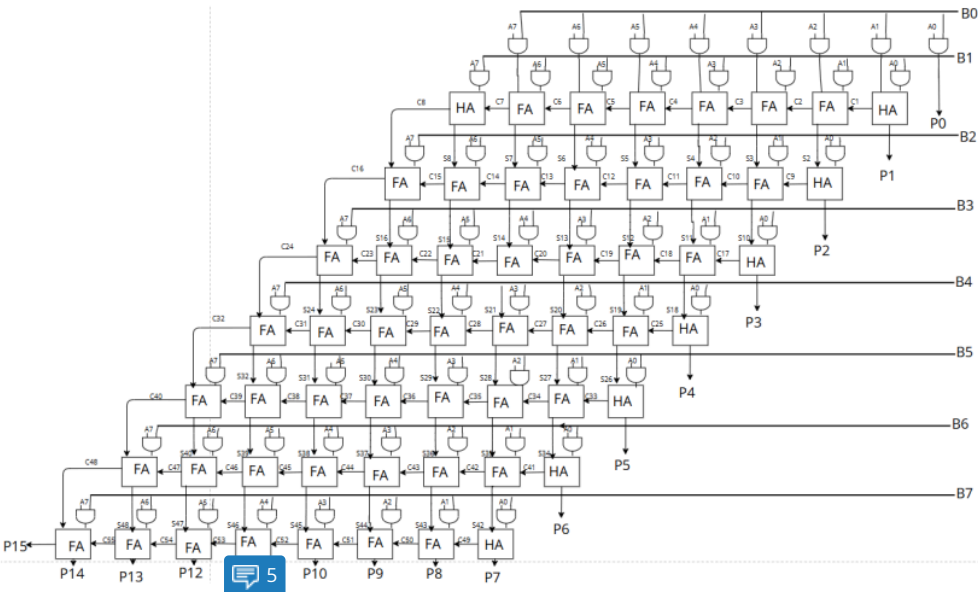


Figure 5

Wallace

Here is the Wallace Tree multiplier design:

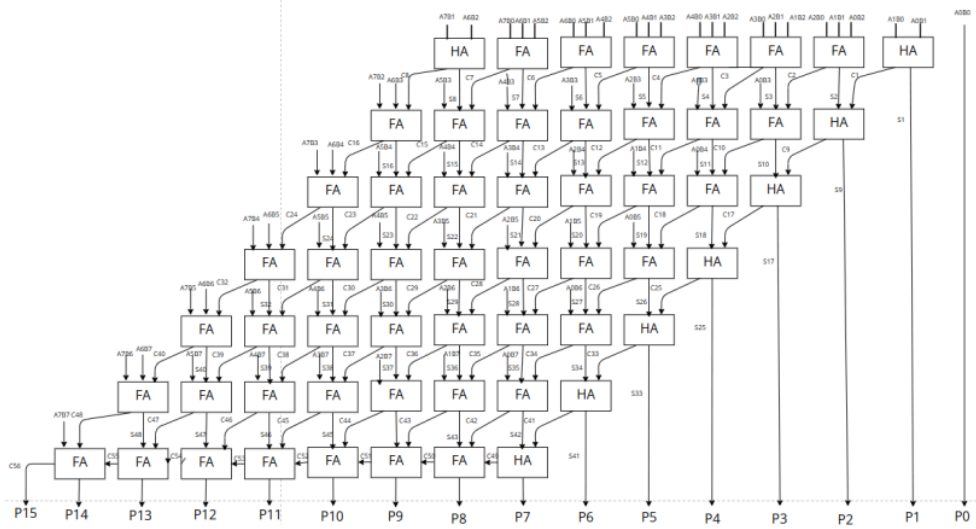


Figure 3

VHDL Code

Brauns

Here is the VHDL code for the Brauns multiplier:

```

1  |
2  | -- Company:
3  | -- Engineer:
4  | --
5  | -- Create Date: 14.04.2025 14:45:34
6  | -- Design Name:
7  | -- Module Name: Brauns 8x8 Main - 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 Brauns_8x8_Main2 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_8x8_Main2;
39 |
40 | architecture Behavioral of Brauns_8x8_Main2 is
41 |     signal A1B0 : std_logic; ---P1 AND GATES
42 |     signal A0B1 : std_logic;
43 |

```

Figure 4

```

44 : signal A2B0 : std_logic; ---P2 AND GATES
45 : signal A1B1 : std_logic;
46 : signal A0B2 : std_logic;
47 :
48 : signal A3B0 : std_logic; ---P3 AND GATES
49 : signal A2B1 : std_logic;
50 : signal A1B2 : std_logic;
51 : signal A0B3 : std_logic;
52 :
53 : signal A4B0 : std_logic; ---P4 AND GATES
54 : signal A3B1 : std_logic;
55 : signal A2B2 : std_logic;
56 : signal A1B3 : std_logic;
57 : signal A0B4 : std_logic;
58 :
59 : signal A5B0 : std_logic; ---P5 AND GATES
60 : signal A4B1 : std_logic;
61 : signal A3B2 : std_logic;
62 : signal A2B3 : std_logic;
63 : signal A1B4 : std_logic;
64 : signal A0B5 : std_logic;
65 :
66 : signal A6B0 : std_logic; ---P6 AND GATES
67 : signal A5B1 : std_logic;
68 : signal A4B2 : std_logic;
69 : signal A3B3 : std_logic;
70 : signal A2B4 : std_logic;
71 : signal A1B5 : std_logic;
72 : signal A0B6 : std_logic;
73 :
74 : signal A7B0 : std_logic; ---P7 AND GATES
75 : signal A6B1 : std_logic;
76 : signal A5B2 : std_logic;
77 : signal A4B3 : std_logic;
78 : signal A3B4 : std_logic;
79 : signal A2B5 : std_logic;
80 : signal A1B6 : std_logic;
81 : signal A0B7 : std_logic;
82 :

```

Figure 5

```

83 : signal A7B1 : std_logic; ---P8 AND GATES
84 : signal A6B2 : std_logic;
85 : signal A5B3 : std_logic;
86 : signal A4B4 : std_logic;
87 : signal A3B5 : std_logic;
88 : signal A2B6 : std_logic;
89 : signal A1B7 : std_logic;
90
91 : signal A7B2 : std_logic; ---P9 AND GATES
92 : signal A6B3 : std_logic;
93 : signal A5B4 : std_logic;
94 : signal A4B5 : std_logic;
95 : signal A3B6 : std_logic;
96 : signal A2B7 : std_logic;
97
98 : signal A7B3 : std_logic; ---P10 AND GATES
99 : signal A6B4 : std_logic;
100 : signal A5B5 : std_logic;
101 : signal A4B6 : std_logic;
102 : signal A3B7 : std_logic;
103
104 : signal A7B4 : std_logic; ---P11 AND GATES
105 : signal A6B5 : std_logic;
106 : signal A5B6 : std_logic;
107 : signal A4B7 : std_logic;
108
109 : signal A7B5 : std_logic; ---P12 AND GATES
110 : signal A6B6 : std_logic;
111 : signal A5B7 : std_logic;
112
113 : signal A7B6 : std_logic; ---P13 AND GATES
114 : signal A6B7 : std_logic;
115
116 : signal A7B7 : std_logic; ---P14 AND GATES

```

Figure 6

```

118 signal C1 : std_logic; ---P2
119 signal S2 : std_logic;
120
121 signal C2 : std_logic; ---P3
122 signal S3 : std_logic;
123 signal C9 : std_logic;
124 signal S10 : std_logic;
125
126 signal C3 : std_logic; ---P4
127 signal S4 : std_logic;
128 signal C10 : std_logic;
129 signal S11 : std_logic;
130 signal C17 : std_logic;
131 signal S18 : std_logic;
132
133 signal C4 : std_logic; ---P5
134 signal S5 : std_logic;
135 signal C11 : std_logic;
136 signal S12 : std_logic;
137 signal C18 : std_logic;
138 signal S19 : std_logic;
139 signal C25 : std_logic;
140 signal S26 : std_logic;
141
142 signal C5 : std_logic; ---P6
143 signal S6 : std_logic;
144 signal C12 : std_logic;
145 signal S13 : std_logic;
146 signal C19 : std_logic;
147 signal S20 : std_logic;
148 signal C26 : std_logic;
149 signal S27 : std_logic;
150 signal C33 : std_logic;
151 signal S34 : std_logic;
--

```

Figure 7

```

153 : signal C6 : std_logic; ---P7
154 : signal S7 : std_logic;
155 : signal C13 : std_logic;
156 : signal S14 : std_logic;
157 : signal C20 : std_logic;
158 : signal S21 : std_logic;
159 : signal C27 : std_logic;
160 : signal S28 : std_logic;
161 : signal C34 : std_logic;
162 : signal S35 : std_logic;
163 : signal C41 : std_logic;
164 : signal S42 : std_logic;
165 :
166 : signal C7 : std_logic; ---P8
167 : signal S8 : std_logic;
168 : signal C14 : std_logic;
169 : signal S15 : std_logic;
170 : signal C21 : std_logic;
171 : signal S22 : std_logic;
172 : signal C28 : std_logic;
173 : signal S29 : std_logic;
174 : signal C35 : std_logic;
175 : signal S36 : std_logic;
176 : signal C42 : std_logic;
177 : signal S43 : std_logic;
178 : signal C49 : std_logic;
179 :
180 : signal C8 : std_logic; ---P9
181 : signal C15 : std_logic;
182 : signal S16 : std_logic;
183 : signal C22 : std_logic;
184 : signal S23 : std_logic;
185 : signal C29 : std_logic;
186 : signal S30 : std_logic;
187 : signal C36 : std_logic;
188 : signal S37 : std_logic;
189 : signal C43 : std_logic;
190 : signal S44 : std_logic;
191 : signal C50 : std_logic;
-- :

```

Figure 8

```

193 signal C16 : std_logic; ---P10
194 signal C23 : std_logic;
195 signal S24 : std_logic;
196 signal C30 : std_logic;
197 signal S31 : std_logic;
198 signal C37 : std_logic;
199 signal S38 : std_logic;
200 signal C44 : std_logic;
201 signal S45 : std_logic;
202 signal C51 : std_logic;
203
204 signal C24 : std_logic; ---P11
205 signal C31 : std_logic;
206 signal S32 : std_logic;
207 signal C38 : std_logic;
208 signal S39 : std_logic;
209 signal C45 : std_logic;
210 signal S46 : std_logic;
211 signal C52 : std_logic;
212
213
214 signal C32 : std_logic; ---P12
215 signal C39 : std_logic;
216 signal S40 : std_logic;
217 signal C46 : std_logic;
218 signal S47 : std_logic;
219 signal C53 : std_logic;
220
221 signal C40 : std_logic; ---P13
222 signal C47 : std_logic;
223 signal S48 : std_logic;
224 signal C54 : std_logic;
225
226 signal C48 : std_logic; ---P14
227 signal C55 : std_logic;
228
229 signal c56 : std_logic; ---P15
230

```

Figure 9

```
231 ⊞ component Full_Adder_Brauns2
232 | port (A : in std_logic;
233 |      B : in std_logic;
234 |      Cin : in std_logic;
235 |      S : out std_logic;
236 |      Cout : out std_logic);
237 ⊞ end component;
238 |
239 ⊞ component Half_Adder_Brauns2
240 | port (A : in std_logic;
241 |      B : in std_logic;
242 |      S : out std_logic;
243 |      Cout : out std_logic);
244 ⊞ end component;
245 |
```

Figure 10

```

247 begin
248 A1B0 <= A(1) and B(0);
249 A0B1 <= A(0) and B(1);
250
251 A2B0 <= A(2) and B(0);
252 A1B1 <= A(1) and B(1);
253 A0B2 <= A(0) and B(2);
254
255 A3B0 <= A(3) and B(0);
256 A2B1 <= A(2) and B(1);
257 A1B2 <= A(1) and B(2);
258 A0B3 <= A(0) and B(3);
259
260 A4B0 <= A(4) and B(0);
261 A3B1 <= A(3) and B(1);
262 A2B2 <= A(2) and B(2);
263 A1B3 <= A(1) and B(3);
264 A0B4 <= A(0) and B(4);
265
266 A5B0 <= A(5) and B(0);
267 A4B1 <= A(4) and B(1);
268 A3B2 <= A(3) and B(2);
269 A2B3 <= A(2) and B(3);
270 A1B4 <= A(1) and B(4);
271 A0B5 <= A(0) and B(5);
272
273 A6B0 <= A(6) and B(0);
274 A5B1 <= A(5) and B(1);
275 A4B2 <= A(4) and B(2);
276 A3B3 <= A(3) and B(3);
277 A2B4 <= A(2) and B(4);
278 A1B5 <= A(1) and B(5);
279 A0B6 <= A(0) and B(6);
280
281 A7B0 <= A(7) and B(0);
282 A6B1 <= A(6) and B(1);
283 A5B2 <= A(5) and B(2);
284 A4B3 <= A(4) and B(3);
285 A3B4 <= A(3) and B(4);
286 A2B5 <= A(2) and B(5);
287 A1B6 <= A(1) and B(6);
288 A0B7 <= A(0) and B(7);

```

Figure 11

```

290 A7B1 <= A(7) and B(1);
291 A6B2 <= A(6) and B(2);
292 A5B3 <= A(5) and B(3);
293 A4B4 <= A(4) and B(4);
294 A3B5 <= A(3) and B(5);
295 A2B6 <= A(2) and B(6);
296 A1B7 <= A(1) and B(7);
297
298 A7B2 <= A(7) and B(2);
299 A6B3 <= A(6) and B(3);
300 A5B4 <= A(5) and B(4);
301 A4B5 <= A(4) and B(5);
302 A3B6 <= A(3) and B(6);
303 A2B7 <= A(2) and B(7);
304
305 A7B3 <= A(7) and B(3);
306 A6B4 <= A(6) and B(4);
307 A5B5 <= A(5) and B(5);
308 A4B6 <= A(4) and B(6);
309 A3B7 <= A(3) and B(7);
310
311 A7B4 <= A(7) and B(4);
312 A6B5 <= A(6) and B(5);
313 A5B6 <= A(5) and B(6);
314 A4B7 <= A(4) and B(7);
315
316 A7B5 <= A(7) and B(5);
317 A6B6 <= A(6) and B(6);
318 A5B7 <= A(5) and B(7);
319
320 A7B6 <= A(7) and B(6);
321 A6B7 <= A(6) and B(7);
322
323 A7B7 <= A(7) and B(7);

```

Figure 12

```

325 P0 : P(0) <= A(0) and B(0);
326
327 P1_0 : Half_Adder_Brauns2 port map (A=>A1B0 , B=>A0B1 , S=> P(1) , Cout=>C1);
328
329 P21 : Full_Adder_Brauns2 port map (A=>A2B0 , B=>A1B1 , Cin=>C1 , S=>S2 , Cout=>C2);
330 P2_0 : Half_Adder_Brauns2 port map (A=>A0B2 , B=>S2 , S=> P(2) , Cout=>C9);
331
332 P31 : Full_Adder_Brauns2 port map (A=>A3B0 , B=>A2B1 , Cin=>C2 , S=>S3 , Cout=>C3);
333 P32 : Full_Adder_Brauns2 port map (A=>A1B2 , B=>S3 , Cin=>C9 , S=>S10 , Cout=>C10);
334 P3_0 : Half_Adder_Brauns2 port map (A=>A0B3 , B=>S10 , S=> P(3) , Cout=>C17);
335
336 P41 : Full_Adder_Brauns2 port map (A=>A4B0 , B=>A3B1 , Cin=>C3 , S=>S4 , Cout=>C4);
337 P42 : Full_Adder_Brauns2 port map (A=>A2B2 , B=>S4 , Cin=>C10 , S=>S11 , Cout=>C11);
338 P43 : Full_Adder_Brauns2 port map (A=>A1B3 , B=>S11 , Cin=>C17 , S=>S18 , Cout=>C18);
339 P4_0 : Half_Adder_Brauns2 port map (A=>A0B4 , B=>S18 , S=> P(4) , Cout=>C25);
340
341 P51 : Full_Adder_Brauns2 port map (A=>A5B0 , B=>A4B1 , Cin=>C4 , S=>S5 , Cout=>C5);
342 P52 : Full_Adder_Brauns2 port map (A=>A3B2 , B=>S5 , Cin=>C11 , S=>S12 , Cout=>C12);
343 P53 : Full_Adder_Brauns2 port map (A=>A2B3 , B=>S12 , Cin=>C18 , S=>S19 , Cout=>C19);
344 P54 : Full_Adder_Brauns2 port map (A=>A1B4 , B=>S19 , Cin=>C25 , S=>S26 , Cout=>C26);
345 P5_0 : Half_Adder_Brauns2 port map (A=>A0B5 , B=>S26 , S=> P(5) , Cout=>C33);
346
347 P61 : Full_Adder_Brauns2 port map (A=>A6B0 , B=>A5B1 , Cin=>C5 , S=>S6 , Cout=>C6);
348 P62 : Full_Adder_Brauns2 port map (A=>A4B2 , B=>S6 , Cin=>C12 , S=>S13 , Cout=>C13);
349 P63 : Full_Adder_Brauns2 port map (A=>A3B3 , B=>S13 , Cin=>C19 , S=>S20 , Cout=>C20);
350 P64 : Full_Adder_Brauns2 port map (A=>A2B4 , B=>S20 , Cin=>C26 , S=>S27 , Cout=>C27);
351 P65 : Full_Adder_Brauns2 port map (A=>A1B5 , B=>S27 , Cin=>C33 , S=>S34 , Cout=>C34);
352 P6_0 : Half_Adder_Brauns2 port map (A=>A0B6 , B=>S34 , S=> P(6) , Cout=>C41);
353
354 P71 : Full_Adder_Brauns2 port map (A=>A7B0 , B=>A6B1 , Cin=>C6 , S=>S7 , Cout=>C7);
355 P72 : Full_Adder_Brauns2 port map (A=>A5B2 , B=>S7 , Cin=>C13 , S=>S14 , Cout=>C14);
356 P73 : Full_Adder_Brauns2 port map (A=>A4B3 , B=>S14 , Cin=>C20 , S=>S21 , Cout=>C21);
357 P74 : Full_Adder_Brauns2 port map (A=>A3B4 , B=>S21 , Cin=>C27 , S=>S28 , Cout=>C28);
358 P75 : Full_Adder_Brauns2 port map (A=>A2B5 , B=>S28 , Cin=>C34 , S=>S35 , Cout=>C35);
359 P76 : Full_Adder_Brauns2 port map (A=>A1B6 , B=>S35 , Cin=>C41 , S=>S42 , Cout=>C42);
360 P7_0 : Half_Adder_Brauns2 port map (A=>A0B7 , B=>S42 , S=> P(7) , Cout=>C49);
361
362 P81 : Half_Adder_Brauns2 port map (A=>A7B1 , B=>C7 , S=>S8 , Cout=>C8);
363 P82 : Full_Adder_Brauns2 port map (A=>A6B2 , B=>S8 , Cin=>C14 , S=>S15 , Cout=>C15);
364 P83 : Full_Adder_Brauns2 port map (A=>A5B3 , B=>S15 , Cin=>C21 , S=>S22 , Cout=>C22);
365 P84 : Full_Adder_Brauns2 port map (A=>A4B4 , B=>S22 , Cin=>C28 , S=>S29 , Cout=>C29);
366 P85 : Full_Adder_Brauns2 port map (A=>A3B5 , B=>S29 , Cin=>C35 , S=>S36 , Cout=>C36);
367 P86 : Full_Adder_Brauns2 port map (A=>A2B6 , B=>S36 , Cin=>C42 , S=>S43 , Cout=>C43);
368 P8_0 : Full_Adder_Brauns2 port map (A=>A1B7 , B=>S43 , Cin=>C49 , S=> P(8) , Cout=>C50);

```

Figure 13

```

370 P91 : Full_Adder_Brauns2 port map (A=>A7B2 , B=>C8 , Cin=>C15 , S=>S16 , Cout=>C16);
371 P92 : Full_Adder_Brauns2 port map (A=>A6B3 , B=>S16 , Cin=>C22 , S=>S23 , Cout=>C23);
372 P93 : Full_Adder_Brauns2 port map (A=>A5B4 , B=>S23 , Cin=>C29 , S=>S30 , Cout=>C30);
373 P94 : Full_Adder_Brauns2 port map (A=>A4B5 , B=>S30 , Cin=>C36 , S=>S37 , Cout=>C37);
374 P95 : Full_Adder_Brauns2 port map (A=>A3B6 , B=>S37 , Cin=>C43 , S=>S44 , Cout=>C44);
375 P9_0 : Full_Adder_Brauns2 port map (A=>A2B7 , B=>S44 , Cin=>C50 , S=> P(9) , Cout=>C51);
376
377 P101 : Full_Adder_Brauns2 port map (A=>A7B3 , B=>C16 , Cin=>C23 , S=>S24 , Cout=>C24);
378 P102 : Full_Adder_Brauns2 port map (A=>A6B4 , B=>S24 , Cin=>C30 , S=>S31 , Cout=>C31);
379 P103 : Full_Adder_Brauns2 port map (A=>A5B5 , B=>S31 , Cin=>C37 , S=>S38 , Cout=>C38);
380 P104 : Full_Adder_Brauns2 port map (A=>A4B6 , B=>S38 , Cin=>C44 , S=>S45 , Cout=>C45);
381 P10_0 : Full_Adder_Brauns2 port map (A=>A3B7 , B=>S45 , Cin=>C51 , S=> P(10) , Cout=>C52);
382
383 P111 : Full_Adder_Brauns2 port map (A=>A7B4 , B=>C24 , Cin=>C31 , S=>S32 , Cout=>C32);
384 P112 : Full_Adder_Brauns2 port map (A=>A6B5 , B=>S32 , Cin=>C38 , S=>S39 , Cout=>C39);
385 P113 : Full_Adder_Brauns2 port map (A=>A5B6 , B=>S39 , Cin=>C45 , S=>S46 , Cout=>C46);
386 P11_0 : Full_Adder_Brauns2 port map (A=>A4B7 , B=>S46 , Cin=>C52 , S=> P(11) , Cout=>C53);
387
388 P121 : Full_Adder_Brauns2 port map (A=>A7B5 , B=>C32 , Cin=>C39 , S=>S40 , Cout=>C40);
389 P122 : Full_Adder_Brauns2 port map (A=>A6B6 , B=>S40 , Cin=>C46 , S=>S47 , Cout=>C47);
390 P12_0 : Full_Adder_Brauns2 port map (A=>A5B7 , B=>S47 , Cin=>C53 , S=> P(12) , Cout=>C54);
391
392 P131 : Full_Adder_Brauns2 port map (A=>A7B6 , B=>C40 , Cin=>C47 , S=>S48 , Cout=>C48);
393 P13_0 : Full_Adder_Brauns2 port map (A=>A6B7 , B=>S48 , Cin=>C54 , S=> P(13) , Cout=>C55);
394
395 P14 : Full_Adder_Brauns2 port map (A=>A7B7 , B=>C48 , Cin=>C55 , S=> P(14) , Cout=>C56);
396
397 P15 : P(15) <=C56;
398
399 end Behavioral;

```

Figure 14

Wallace

Here is the VHDL code for the Wallace Tree multiplier:

```
1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 14.04.2025 19:26:34
6  -- Design Name:
7  -- Module Name: Wallace 8x8 Main - 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
```

Figure 15

```

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_8x8_Main 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 Wallace_8x8_Main;
39 |
40 | architecture Behavioral of Wallace_8x8_Main is
41 |
42 | component Full_Adder_Wallace is
43 |     Port ( A : in  STD_LOGIC;
44 |           B : in  STD_LOGIC;
45 |           Cin : in  STD_LOGIC;
46 |           S : out STD_LOGIC;
47 |           Cout : out STD_LOGIC);
48 | end component;
49 |
50 | component Half_Adder_Wallace is
51 |     Port ( A : in  STD_LOGIC;
52 |           B : in  STD_LOGIC;
53 |           S : out STD_LOGIC;
54 |           Cout : out STD_LOGIC);
55 | end component;

```

Figure 16

```

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

```

Figure 17

```

60 signal A1B0 : std_logic;
61 signal A0B1 : std_logic;
62 signal A2B0 : std_logic;
63 signal A1B1 : std_logic;
64 signal A0B2 : std_logic;
65 signal A3B0 : std_logic;
66 signal A2B1 : std_logic;
67 signal A1B2 : std_logic;
68 signal A0B3 : std_logic;
69 signal A4B0 : std_logic;
70 signal A3B1 : std_logic;
71 signal A2B2 : std_logic;
72 signal A1B3 : std_logic;
73 signal A0B4 : std_logic;
74 signal A5B0 : std_logic;
75 signal A4B1 : std_logic;
76 signal A3B2 : std_logic;
77 signal A2B3 : std_logic;
78 signal A1B4 : std_logic;
79 signal A0B5 : std_logic;
80 signal A6B0 : std_logic;
81 signal A5B1 : std_logic;
82 signal A4B2 : std_logic;
83 signal A3B3 : std_logic;
84 signal A2B4 : std_logic;
85 signal A1B5 : std_logic;
86 signal A0B6 : std_logic;
87 signal A7B0 : std_logic;
88 signal A6B1 : std_logic;
89 signal A5B2 : std_logic;
90 signal A4B3 : std_logic;
91 signal A3B4 : std_logic;
92 signal A2B5 : std_logic;
93 signal A1B6 : std_logic;
94 signal A0B7 : std_logic;
95 signal A7B1 : std_logic;
96 signal A6B2 : std_logic;
97 signal A5B3 : std_logic;
98 signal A4B4 : std_logic;
99 signal A3B5 : std_logic;
100 signal A2B6 : std_logic;
101 signal A1B7 : std_logic;
102 signal A7B2 : std_logic;
103 signal A6B3 : std_logic;

```

Figure 18

```
104 : signal A5B4 : std_logic;
105 : signal A4B5 : std_logic;
106 : signal A3B6 : std_logic;
107 : signal A2B7 : std_logic;
108 : signal A7B3 : std_logic;
109 : signal A6B4 : std_logic;
110 : signal A5B5 : std_logic;
111 : signal A4B6 : std_logic;
112 : signal A3B7 : std_logic;
113 : signal A7B4 : std_logic;
114 : signal A6B5 : std_logic;
115 : signal A5B6 : std_logic;
116 : signal A4B7 : std_logic;
117 : signal A7B5 : std_logic;
118 : signal A6B6 : std_logic;
119 : signal A5B7 : std_logic;
120 : signal A7B6 : std_logic;
121 : signal A6B7 : std_logic;
122 : signal A7B7 : std_logic;
```

Figure 19

```

124   begin
125
126   A1B0 <= A(1) and B(0);
127   A0B1 <= A(0) and B(1);
128
129   A2B0 <= A(2) and B(0);
130   A1B1 <= A(1) and B(1);
131   A0B2 <= A(0) and B(2);
132
133   A3B0 <= A(3) and B(0);
134   A2B1 <= A(2) and B(1);
135   A1B2 <= A(1) and B(2);
136   A0B3 <= A(0) and B(3);
137
138   A4B0 <= A(4) and B(0);
139   A3B1 <= A(3) and B(1);
140   A2B2 <= A(2) and B(2);
141   A1B3 <= A(1) and B(3);
142   A0B4 <= A(0) and B(4);
143
144   A5B0 <= A(5) and B(0);
145   A4B1 <= A(4) and B(1);
146   A3B2 <= A(3) and B(2);
147   A2B3 <= A(2) and B(3);
148   A1B4 <= A(1) and B(4);
149   A0B5 <= A(0) and B(5);
150
151   A6B0 <= A(6) and B(0);
152   A5B1 <= A(5) and B(1);
153   A4B2 <= A(4) and B(2);
154   A3B3 <= A(3) and B(3);
155   A2B4 <= A(2) and B(4);
156   A1B5 <= A(1) and B(5);
157   A0B6 <= A(0) and B(6);
158
159   A7B0 <= A(7) and B(0);
160   A6B1 <= A(6) and B(1);
161   A5B2 <= A(5) and B(2);
162   A4B3 <= A(4) and B(3);
163   A3B4 <= A(3) and B(4);
164   A2B5 <= A(2) and B(5);
165   A1B6 <= A(1) and B(6);
166   A0B7 <= A(0) and B(7);

```

Figure 20

```

168 : A7B1 <= A(7) and B(1);
169 : A6B2 <= A(6) and B(2);
170 : A5B3 <= A(5) and B(3);
171 : A4B4 <= A(4) and B(4);
172 : A3B5 <= A(3) and B(5);
173 : A2B6 <= A(2) and B(6);
174 : A1B7 <= A(1) and B(7);
175 :
176 : A7B2 <= A(7) and B(2);
177 : A6B3 <= A(6) and B(3);
178 : A5B4 <= A(5) and B(4);
179 : A4B5 <= A(4) and B(5);
180 : A3B6 <= A(3) and B(6);
181 : A2B7 <= A(2) and B(7);
182 :
183 : A7B3 <= A(7) and B(3);
184 : A6B4 <= A(6) and B(4);
185 : A5B5 <= A(5) and B(5);
186 : A4B6 <= A(4) and B(6);
187 : A3B7 <= A(3) and B(7);
188 :
189 : A7B4 <= A(7) and B(4);
190 : A6B5 <= A(6) and B(5);
191 : A5B6 <= A(5) and B(6);
192 : A4B7 <= A(4) and B(7);
193 :
194 : A7B5 <= A(7) and B(5);
195 : A6B6 <= A(6) and B(6);
196 : A5B7 <= A(5) and B(7);
197 :
198 : A7B6 <= A(7) and B(6);
199 : A6B7 <= A(6) and B(7);
200 :
201 : A7B7 <= A(7) and B(7);

```

Figure 21

```

203 ---First Stage
204 P0 : P(0) <= A(0) and B(0);
205
206 ha11 : Half_Adder_Wallace port map (A=>A1B0,B=>A0B1,S=>P(1),Cout=>C1);
207 fa12 : Full_Adder_Wallace port map (A=>A2B0,B=>A1B1,Cin=>A0B2,S=>S2,Cout=>C2);
208 fa13 : Full_Adder_Wallace port map (A=>A3B0,B=>A2B1,Cin=>A1B2,S=>S3,Cout=>C3);
209 fa14 : Full_Adder_Wallace port map (A=>A4B0,B=>A3B1,Cin=>A2B2,S=>S4,Cout=>C4);
210 fa15 : Full_Adder_Wallace port map (A=>A5B0,B=>A4B1,Cin=>A3B2,S=>S5,Cout=>C5);
211 fa16 : Full_Adder_Wallace port map (A=>A6B0,B=>A5B1,Cin=>A4B2,S=>S6,Cout=>C6);
212 fa17 : Full_Adder_Wallace port map (A=>A7B0,B=>A6B1,Cin=>A5B2,S=>S7,Cout=>C7);
213 ha18 : Half_Adder_Wallace port map (A=>A7B1,B=>A6B2,S=>S8,Cout=>C8);
214
215 ---Second Stage
216 ha21 : Half_Adder_Wallace port map (A=>S2,B=>C1,S=>P(2),Cout=>C9);
217 fa22 : Full_Adder_Wallace port map (A=>A0B3,B=>S3,Cin=>C2,S=>S10,Cout=>C10);
218 fa23 : Full_Adder_Wallace port map (A=>A1B3,B=>S4,Cin=>C3,S=>S11,Cout=>C11);
219 fa24 : Full_Adder_Wallace port map (A=>A2B3,B=>S5,Cin=>C4,S=>S12,Cout=>C12);
220 fa25 : Full_Adder_Wallace port map (A=>A3B3,B=>S6,Cin=>C5,S=>S13,Cout=>C13);
221 fa26 : Full_Adder_Wallace port map (A=>A4B3,B=>S7,Cin=>C6,S=>S14,Cout=>C14);
222 fa27 : Full_Adder_Wallace port map (A=>A5B3,B=>S8,Cin=>C7,S=>S15,Cout=>C15);
223 fa28 : Full_Adder_Wallace port map (A=>A6B3,B=>A7B2,Cin=>C8,S=>S16,Cout=>C16);
224
225 ---Third Stage
226 ha31 : Half_Adder_Wallace port map (A=>S10,B=>C9,S=>P(3),Cout=>C17);
227 fa32 : Full_Adder_Wallace port map (A=>A0B4,B=>S11,Cin=>C10,S=>S18,Cout=>C18);
228 fa33 : Full_Adder_Wallace port map (A=>A1B4,B=>S12,Cin=>C11,S=>S19,Cout=>C19);
229 fa34 : Full_Adder_Wallace port map (A=>A2B4,B=>S13,Cin=>C12,S=>S20,Cout=>C20);
230 fa35 : Full_Adder_Wallace port map (A=>A3B4,B=>S14,Cin=>C13,S=>S21,Cout=>C21);
231 fa36 : Full_Adder_Wallace port map (A=>A4B4,B=>S15,Cin=>C14,S=>S22,Cout=>C22);
232 fa37 : Full_Adder_Wallace port map (A=>A5B4,B=>S16,Cin=>C15,S=>S23,Cout=>C23);
233 fa38 : Full_Adder_Wallace port map (A=>A6B4,B=>A7B3,Cin=>C16,S=>S24,Cout=>C24);
234
235 ---Fourth Stage
236 ha41 : Half_Adder_Wallace port map (A=>S18,B=>C17,S=>P(4),Cout=>C25);
237 fa42 : Full_Adder_Wallace port map (A=>A0B5,B=>S19,Cin=>C18,S=>S26,Cout=>C26);
238 fa43 : Full_Adder_Wallace port map (A=>A1B5,B=>S20,Cin=>C19,S=>S27,Cout=>C27);
239 fa44 : Full_Adder_Wallace port map (A=>A2B5,B=>S21,Cin=>C20,S=>S28,Cout=>C28);
240 fa45 : Full_Adder_Wallace port map (A=>A3B5,B=>S22,Cin=>C21,S=>S29,Cout=>C29);
241 fa46 : Full_Adder_Wallace port map (A=>A4B5,B=>S23,Cin=>C22,S=>S30,Cout=>C30);
242 fa47 : Full_Adder_Wallace port map (A=>A5B5,B=>S24,Cin=>C23,S=>S31,Cout=>C31);
243 fa48 : Full_Adder_Wallace port map (A=>A6B5,B=>A7B4,Cin=>C24,S=>S32,Cout=>C32);

```

Figure 22

```

245 : ---Fifth Stage
246 : ha51 : Half_Adder_Wallace port map (A=>S26,B=>C25,S=>P(5),Cout=>C33);
247 : fa52 : Full_Adder_Wallace port map (A=>A0B6,B=>S27,Cin=>C26,S=>S34,Cout=>C34);
248 : fa53 : Full_Adder_Wallace port map (A=>A1B6,B=>S28,Cin=>C27,S=>S35,Cout=>C35);
249 : fa54 : Full_Adder_Wallace port map (A=>A2B6,B=>S29,Cin=>C28,S=>S36,Cout=>C36);
250 : fa55 : Full_Adder_Wallace port map (A=>A3B6,B=>S30,Cin=>C29,S=>S37,Cout=>C37);
251 : fa56 : Full_Adder_Wallace port map (A=>A4B6,B=>S31,Cin=>C30,S=>S38,Cout=>C38);
252 : fa57 : Full_Adder_Wallace port map (A=>A5B6,B=>S32,Cin=>C31,S=>S39,Cout=>C39);
253 : fa58 : Full_Adder_Wallace port map (A=>A7B5,B=>A6B6,Cin=>C32,S=>S40,Cout=>C40);
254 :
255 : ---Sixth Stage
256 : ha61 : Half_Adder_Wallace port map (A=>S34,B=>C33,S=>P(6),Cout=>C41);
257 : fa62 : Full_Adder_Wallace port map (A=>A0B7,B=>S35,Cin=>C34,S=>S42,Cout=>C42);
258 : fa63 : Full_Adder_Wallace port map (A=>A1B7,B=>S36,Cin=>C35,S=>S43,Cout=>C43);
259 : fa64 : Full_Adder_Wallace port map (A=>A2B7,B=>S37,Cin=>C36,S=>S44,Cout=>C44);
260 : fa65 : Full_Adder_Wallace port map (A=>A3B7,B=>S38,Cin=>C37,S=>S45,Cout=>C45);
261 : fa66 : Full_Adder_Wallace port map (A=>A4B7,B=>S39,Cin=>C38,S=>S46,Cout=>C46);
262 : fa67 : Full_Adder_Wallace port map (A=>A5B7,B=>S40,Cin=>C39,S=>S47,Cout=>C47);
263 : fa68 : Full_Adder_Wallace port map (A=>A7B6,B=>A6B7,Cin=>C40,S=>S48,Cout=>C48);
264 :
265 : ---Seventh Stage
266 : ha71 : Half_Adder_Wallace port map (A=>S42,B=>C41,S=>P(7),Cout=>C49);
267 : fa72 : Full_Adder_Wallace port map (A=>S43,B=>C42,Cin=>C49,S=>P(8),Cout=>C50);
268 : fa73 : Full_Adder_Wallace port map (A=>S44,B=>C43,Cin=>C50,S=>P(9),Cout=>C51);
269 : fa74 : Full_Adder_Wallace port map (A=>S45,B=>C44,Cin=>C51,S=>P(10),Cout=>C52);
270 : fa75 : Full_Adder_Wallace port map (A=>S46,B=>C45,Cin=>C52,S=>P(11),Cout=>C53);
271 : fa76 : Full_Adder_Wallace port map (A=>S47,B=>C46,Cin=>C53,S=>P(12),Cout=>C54);
272 : fa77 : Full_Adder_Wallace port map (A=>S48,B=>C47,Cin=>C54,S=>P(13),Cout=>C55);
273 : fa78 : Full_Adder_Wallace port map (A=>A7B7,B=>C48,Cin=>C55,S=>P(14),Cout=>C56);
274 :
275 : P15 : P(15) <=C56;
276 :
277 : end Behavioral;

```

Figure 23

Timings & Schematics

Brauns

Here is the timing constraints code for the Brauns multiplier:

```

67 create_clock -period 41.000 -name virtual_clk -waveform {0.000 20.500}
68
69 # Set delays based on analytical calculations
70 set_input_delay -clock virtual_clk -max 1.000 [get_ports A*]
71 set_input_delay -clock virtual_clk -max 1.000 [get_ports B*]
72 set_output_delay -clock virtual_clk -max 2.000 [get_ports P*]
73
74 # Set max delay for the critical path (longest delay)
75 set_max_delay -from [get_ports A*] -to [get_ports P*] 41.000
76
77 set_operating_conditions -design_power_budget 2.0
78
79 set_operating_conditions -design_power_budget 2.0

```

Figure 24

Here is the schematic from implementation for the Brauns multiplier

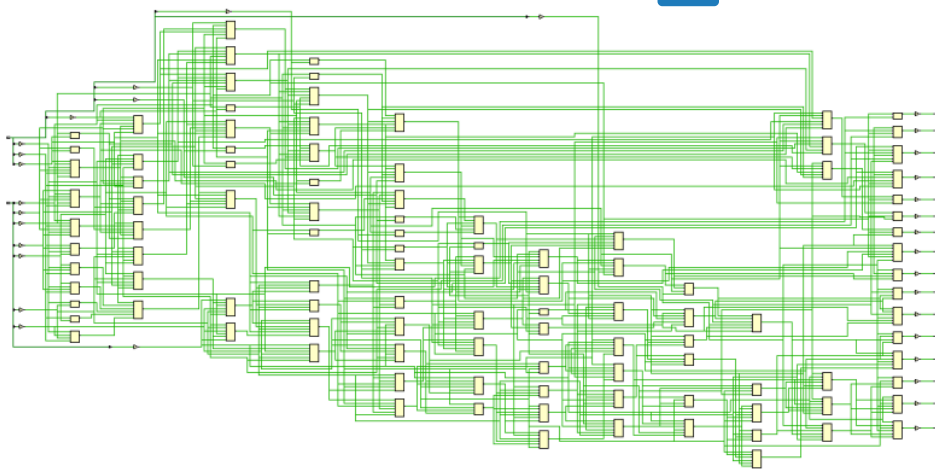


Figure 25

Wallace

Here is the timing constraints code for the Wallace Tree multiplier:

```

1: create_clock -period 29.000 -name virtual_clk -waveform {0.000 14.500}
2:
3: # Set delays based on analytical calculations
4: set_input_delay -clock virtual_clk -max 1.000 [get_ports A*]
5: set_input_delay -clock virtual_clk -max 1.000 [get_ports B*]
6: set_output_delay -clock virtual_clk -max 2.000 [get_ports P*]
7:
8: # Set max delay for the critical path (longest delay)
9: set_max_delay -from [get_ports A*] -to [get_ports P*] 29.000
10:
11: set_operating_conditions -design_power_budget 2.0
12:
13: set_property IOSTANDARD LVCOS33 [get_ports {P[2]}]
14: set_property IOSTANDARD LVCOS33 [get_ports {P[1]}]
15: set_property IOSTANDARD LVCOS33 [get_ports {P[0]}]
16: set_property IOSTANDARD LVCOS33 [get_ports {P[3]}]
17: set_property IOSTANDARD LVCOS33 [get_ports {P[4]}]
18: set_property IOSTANDARD LVCOS33 [get_ports {P[5]}]
19: set_property IOSTANDARD LVCOS33 [get_ports {P[6]}]
20: set_property IOSTANDARD LVCOS33 [get_ports {A[7]}]
21: set_property IOSTANDARD LVCOS33 [get_ports {A[6]}]
22: set_property IOSTANDARD LVCOS33 [get_ports {A[5]}]
23: set_property IOSTANDARD LVCOS33 [get_ports {A[4]}]
24: set_property IOSTANDARD LVCOS33 [get_ports {A[3]}]
25: set_property IOSTANDARD LVCOS33 [get_ports {A[2]}]
26: set_property IOSTANDARD LVCOS33 [get_ports {A[1]}]
27: set_property IOSTANDARD LVCOS33 [get_ports {A[0]}]
28: set_property IOSTANDARD LVCOS33 [get_ports {B[7]}]

```

Figure 26

```

29: set_property IOSTANDARD LVCOS33 [get_ports {B[6]}]
30: set_property IOSTANDARD LVCOS33 [get_ports {B[5]}]
31: set_property IOSTANDARD LVCOS33 [get_ports {B[4]}]
32: set_property IOSTANDARD LVCOS33 [get_ports {B[2]}]
33: set_property IOSTANDARD LVCOS33 [get_ports {B[3]}]
34: set_property IOSTANDARD LVCOS33 [get_ports {B[1]}]
35: set_property IOSTANDARD LVCOS33 [get_ports {B[0]}]
36: set_property IOSTANDARD LVCOS33 [get_ports {P[15]}]
37: set_property IOSTANDARD LVCOS33 [get_ports {P[14]}]
38: set_property IOSTANDARD LVCOS33 [get_ports {P[13]}]
39: set_property IOSTANDARD LVCOS33 [get_ports {P[12]}]
40: set_property IOSTANDARD LVCOS33 [get_ports {P[11]}]
41: set_property IOSTANDARD LVCOS33 [get_ports {P[10]}]
42: set_property IOSTANDARD LVCOS33 [get_ports {P[9]}]
43: set_property IOSTANDARD LVCOS33 [get_ports {P[8]}]
44: set_property IOSTANDARD LVCOS33 [get_ports {P[7]}]
45:
46: create_clock -period 29.000 -name Wallace_Clock -waveform {0.000 14.500}

```

Figure 27

Here is the schematic from implementation for the Wallace Tree multiplier:

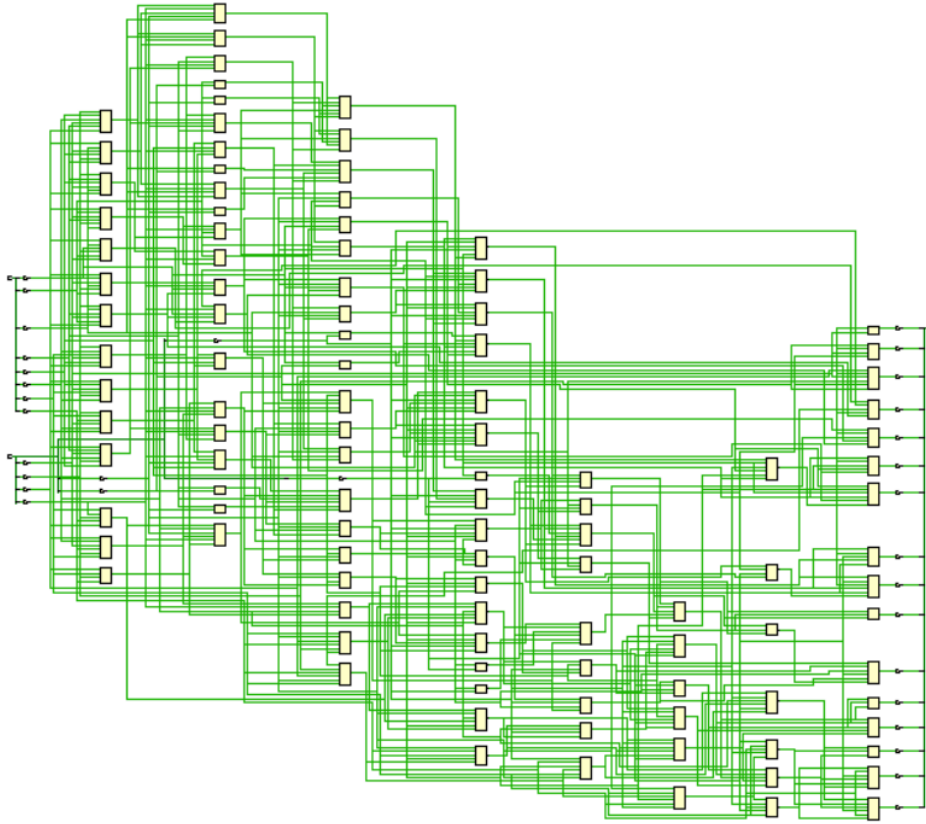


Figure 28

FPGA Design & Pin Allocation

Brauns

Here is the RTL analysis of the Brauns multiplier:

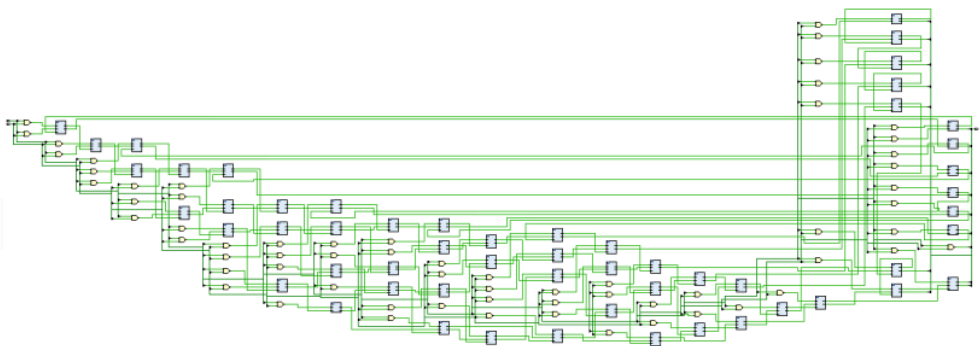


Figure 29

Here is the pin allocation of the Brauns multiplier:

Tcl Console Messages Log Reports Design Items Find Results x Timing																
Name	Direction	Interface	Reg Diff Par	Package Pin	Fixed	Bank	I/O Std	Vccio	Vref	Drive Strength	Slew Type	Pull Type	Off-Chip Termination	IN_TERM	Partition Pin Location	
A[7]	IN			R2	✓	34	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
A[6]	IN			T1	✓	34	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
A[5]	IN			U1	✓	34	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
A[4]	IN			W2	✓	34	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
A[3]	IN			R3	✓	34	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
A[2]	IN			T2	✓	34	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
A[1]	IN			T3	✓	34	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
A[0]	IN			V2	✓	34	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
B[7]	IN			W13	✓	14	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
B[6]	IN			W14	✓	14	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
B[5]	IN			V15	✓	14	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
B[4]	IN			W15	✓	14	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
B[3]	IN			W17	✓	14	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
B[2]	IN			W16	✓	14	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
B[1]	IN			V16	✓	14	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
B[0]	IN			V17	✓	14	LVCMOS33*	3.300				NONE	✓	NONE	✓	N/A
P[15]	OUT			L1	✓	35	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[14]	OUT			P1	✓	35	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[13]	OUT			N3	✓	35	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[12]	OUT			P3	✓	35	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[11]	OUT			U3	✓	34	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[10]	OUT			W3	✓	34	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[9]	OUT			V3	✓	34	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[8]	OUT			V13	✓	14	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[7]	OUT			V14	✓	14	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[6]	OUT			U14	✓	14	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[5]	OUT			U15	✓	14	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[4]	OUT			W18	✓	14	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[3]	OUT			V19	✓	14	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[2]	OUT			U19	✓	14	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[1]	OUT			E19	✓	14	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A
P[0]	OUT			U16	✓	14	LVCMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	✓	N/A

Figure 30

Here is the device view from implementation for the Brauns multiplier:

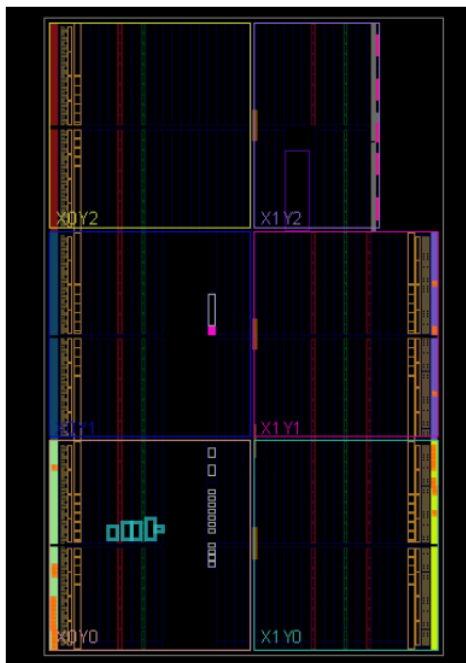


Figure 31

Here is the I/O planning for the Brauns multiplier:

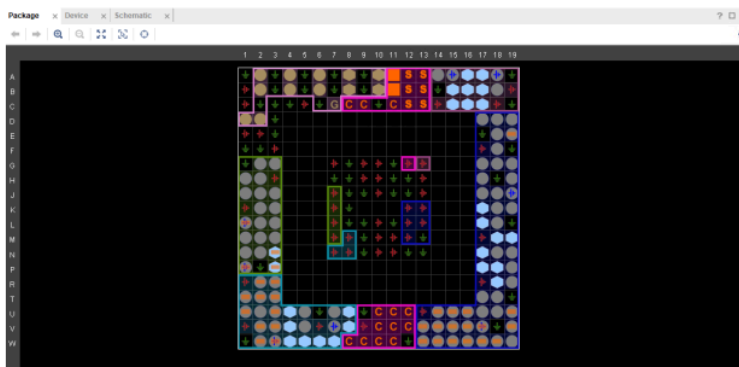


Figure 32

Wallace

Here is the RTL analysis of the Wallace Tree multiplier:

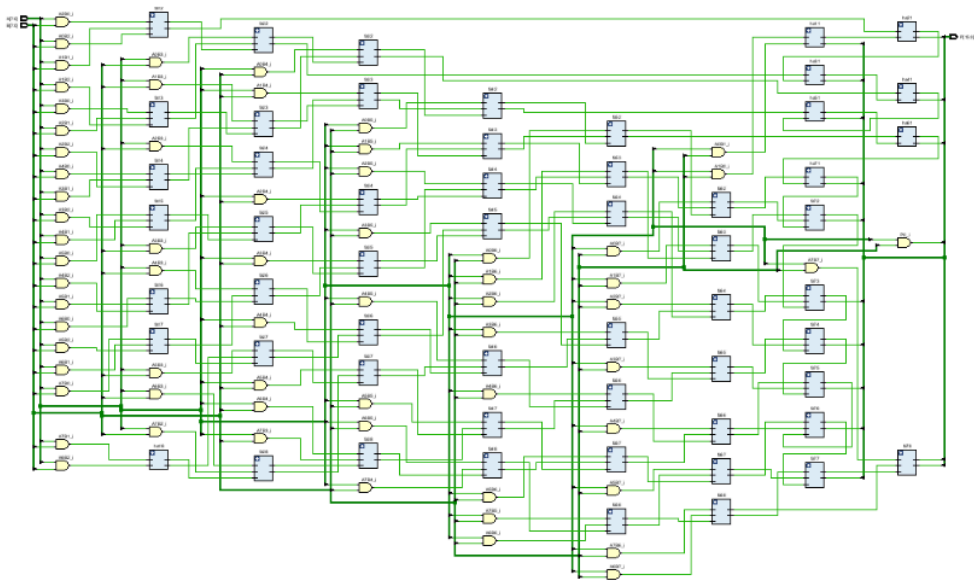


Figure 33

Here is the pin allocation of the Wallace Tree multiplier:

Name	Direction	Interface	Net Diff Pair	Package Pin	Fixed	Bank	I/O Std	VCCA	Vref	Drive Strength	Stee Type	Pull Type	OR-Clp Termination	PL_TERM	Partition Pin Location
AP1	IN			R2	✓	34	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP2	IN			T1	✓	34	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP3	IN			U1	✓	34	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP4	IN			W2	✓	34	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP5	IN			R3	✓	34	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP6	IN			T2	✓	34	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP7	IN			T3	✓	34	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP8	IN			V2	✓	34	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP9	IN			W13	✓	14	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP10	IN			W14	✓	14	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP11	IN			V15	✓	14	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP12	IN			W15	✓	14	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP13	IN			W17	✓	14	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP14	IN			W16	✓	14	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP15	IN			V16	✓	14	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
AP16	IN			V17	✓	14	LVCNMOS33*	3.300				NONE	✓	NONE	N/A
OP15	OUT			L1	✓	35	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP14	OUT			P1	✓	35	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP13	OUT			N3	✓	35	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP12	OUT			P3	✓	35	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP11	OUT			U3	✓	34	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP10	OUT			W3	✓	34	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP9	OUT			V3	✓	34	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP8	OUT			V13	✓	14	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP7	OUT			V14	✓	14	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP6	OUT			U14	✓	14	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP5	OUT			U15	✓	14	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP4	OUT			W16	✓	14	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP3	OUT			V19	✓	14	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP2	OUT			U19	✓	14	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP1	OUT			E19	✓	14	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A
OP0	OUT			U16	✓	14	LVCNMOS33*	3.300	12	✓	✓	NONE	✓	FP_VTT_50	N/A

Figure 34

Here is the device view from implementation for the Wallace Tree multiplier:

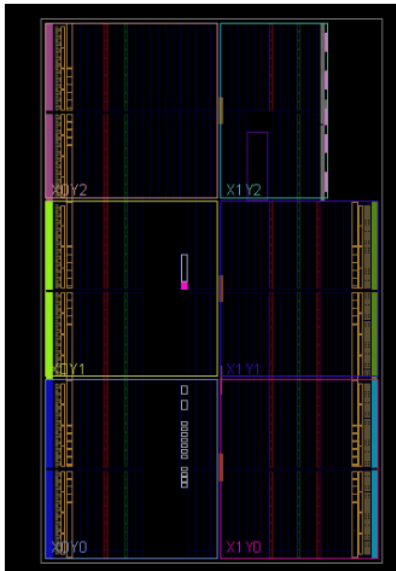


Figure 35

Here is the I/O planning for the Wallace Tree multiplier:

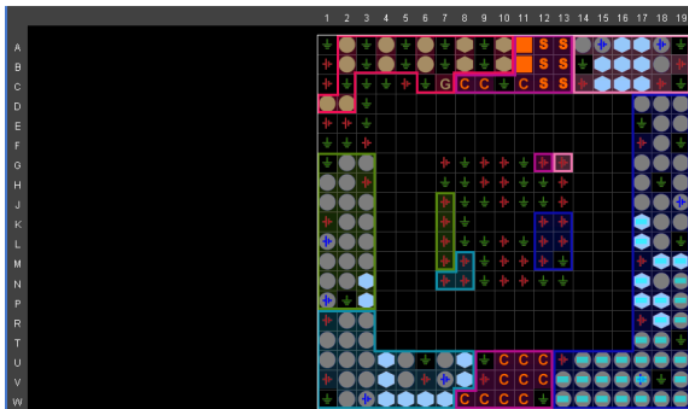


Figure 36

Simulation Results

Brauns

Here are the four example calculations inputted into the Brauns multiplier:

EF x FF:

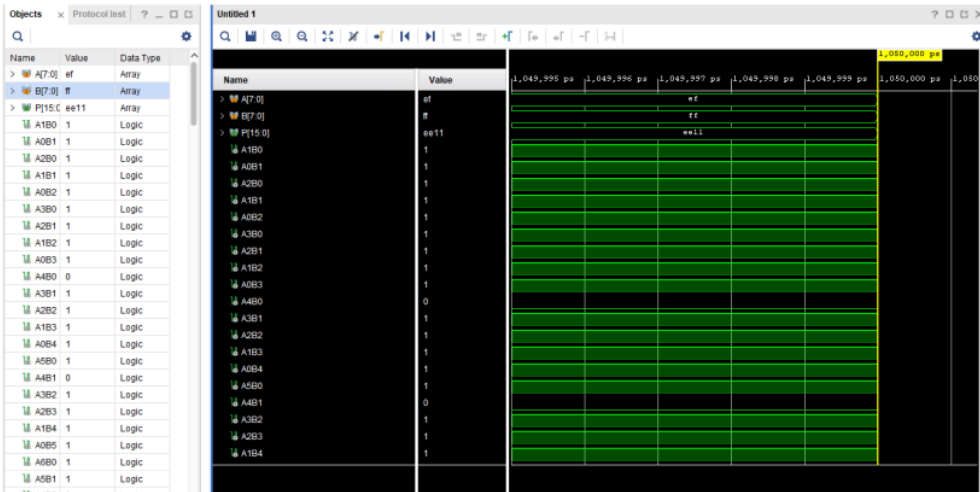


Figure 37

B4 x C9:

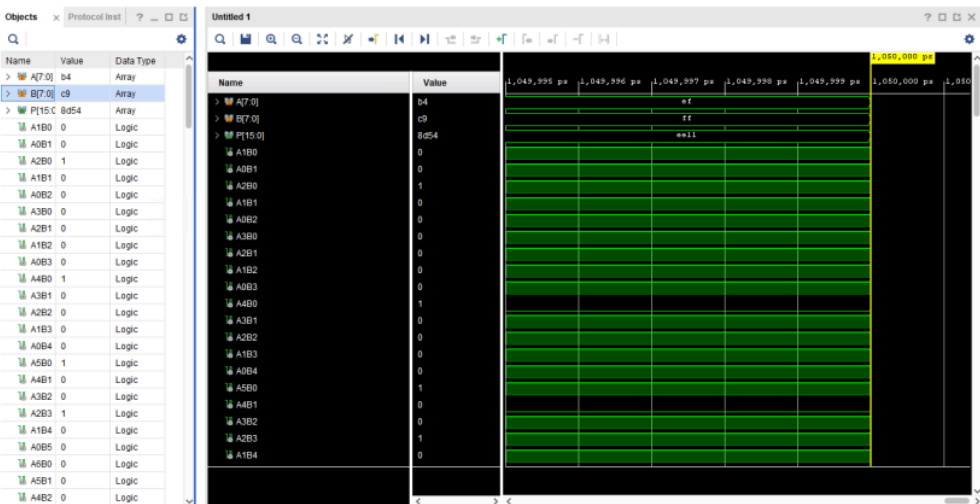


Figure 38

The screenshot shows the GDB interface with two windows open: 'Objects' and 'Registers'.

Objects Window:

Name	Value	Data Type
B[7:0]	85	Array
B[7:0]	a1	Array
P[15:0]	9ab6	Array
A1B0	1	Logic
A0B1	0	Logic
A2B0	1	Logic
A1B1	0	Logic
A0B2	0	Logic
A3B0	0	Logic
A2B1	0	Logic
A1B2	0	Logic
A0B3	0	Logic
A4B0	1	Logic
A3B1	0	Logic
A2B2	0	Logic
A1B3	0	Logic
A0B4	0	Logic
A5B0	1	Logic
A4B1	0	Logic
A3B2	0	Logic
A2B3	0	Logic
A1B4	0	Logic
A0B5	0	Logic
A6B0	1	Logic
A5B1	0	Logic

Registers Window:

Name	Value
RAX	1,049,995 px
RDX	1,049,996 px
RDI	1,049,997 px
RSI	1,049,998 px
RBP	1,049,999 px
RSP	1,050,000 px
R8	1,050,001 px
R9	1,050,002 px
R10	1,050,003 px
R11	1,050,004 px
R12	1,050,005 px
R13	1,050,006 px
R14	1,050,007 px
R15	1,050,008 px
CR2	1,050,009 px
CR3	1,050,010 px
CR4	1,050,011 px
CR8	1,050,012 px

Figure 39

The screenshot shows the 'Objects' panel in a debugger. On the left, a list of objects is displayed with columns for Name, Value, and Data Type. The object 'P15C' with value '4050' and data type 'Array' is selected. On the right, a detailed view of the 'P15C' object is shown, displaying its internal structure as an array of 16 elements. The first element is '62', and the rest are '0'. The array is named 'P15C' and has a value of '4050'.

Name	Value	Data Type
> A7[0]	62	Array
> B7[0]	a8	Array
> P15C	4050	Array
A1B0	0	Logic
A0B1	0	Logic
A2B0	0	Logic
A1B1	0	Logic
A0B2	0	Logic
A3B0	0	Logic
A2B1	0	Logic
A1B2	0	Logic
A0B3	0	Logic
A4B0	0	Logic
A3B1	0	Logic
A2B2	0	Logic
A1B3	1	Logic
A0B4	0	Logic
A5B0	0	Logic
A4B1	0	Logic
A3B2	0	Logic
A2B3	0	Logic
A1B4	0	Logic
A0B5	0	Logic
A6B0	0	Logic
A5B1	0	Logic

Name	Value
> A7[0]	62
> B7[0]	a8
> P15[0]	4050
A1B0	0
A0B1	0
A2B0	0
A1B1	0
A0B2	0
A3B0	0
A2B1	0
A1B2	0
A0B3	0
A4B0	0
A3B1	0
A2B2	0
A1B3	1
A0B4	0
A5B0	0
A4B1	0
A3B2	0
A2B3	0
A1B4	0

Figure 40

Here is the timing summary for the Brauns multiplier:

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 13.508 ns	Worst Hold Slack (WHS): NA	Worst Pulse Width Slack (WPWS): NA
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): NA	Total Pulse Width Negative Slack (TPWS): NA
Number of Failing Endpoints: 0	Number of Failing Endpoints: NA	Number of Failing Endpoints: NA
Total Number of Endpoints: 16	Total Number of Endpoints: NA	Total Number of Endpoints: NA

All user specified timing constraints are met.

Figure 41

Here is the combinational delay for the Brauns multiplier:

Combinational Delays

From Port	To Port	Max Delay	Max Process Corner
A[0]	P[0]	11.767	SLOW
A[0]	P[1]	11.279	SLOW
A[0]	P[2]	10.848	SLOW
A[0]	P[3]	11.307	SLOW
A[0]	P[4]	14.084	SLOW
A[0]	P[5]	14.607	SLOW
A[0]	P[6]	16.557	SLOW
A[0]	P[7]	16.445	SLOW
A[0]	P[8]	17.993	SLOW
A[0]	P[9]	20.129	SLOW
A[0]	P[10]	20.276	SLOW
A[0]	P[11]	21.425	SLOW
A[0]	P[12]	22.391	SLOW
A[0]	P[13]	22.818	SLOW
A[0]	P[14]	23.219	SLOW
A[0]	P[15]	23.765	SLOW
A[1]	P[1]	11.295	SLOW
A[1]	P[2]	11.271	SLOW
A[1]	P[3]	11.767	SLOW
A[1]	P[4]	14.440	SLOW
A[1]	P[5]	15.143	SLOW
A[1]	P[6]	16.913	SLOW
A[1]	P[7]	16.920	SLOW
A[1]	P[8]	18.371	SLOW
A[1]	P[9]	20.486	SLOW
A[1]	P[10]	20.632	SLOW
A[1]	P[11]	21.781	SLOW
A[1]	P[12]	22.747	SLOW
A[1]	P[13]	23.174	SLOW
A[1]	P[14]	23.576	SLOW
A[1]	P[15]	24.121	SLOW
A[2]	P[2]	11.610	SLOW
A[2]	P[3]	11.529	SLOW

Figure 42

👉 A[2]	👉 P[4]	13.974	SLOW
👉 A[2]	👉 P[5]	14.828	SLOW
👉 A[2]	👉 P[6]	16.447	SLOW
👉 A[2]	👉 P[7]	16.605	SLOW
👉 A[2]	👉 P[8]	18.056	SLOW
👉 A[2]	👉 P[9]	20.020	SLOW
👉 A[2]	👉 P[10]	20.166	SLOW
👉 A[2]	👉 P[11]	21.315	SLOW
👉 A[2]	👉 P[12]	22.281	SLOW
👉 A[2]	👉 P[13]	22.708	SLOW
👉 A[2]	👉 P[14]	23.110	SLOW
👉 A[2]	👉 P[15]	23.655	SLOW
👉 A[3]	👉 P[3]	11.160	SLOW
👉 A[3]	👉 P[4]	14.786	SLOW
👉 A[3]	👉 P[5]	15.039	SLOW
👉 A[3]	👉 P[6]	17.259	SLOW
👉 A[3]	👉 P[7]	17.148	SLOW
👉 A[3]	👉 P[8]	18.695	SLOW
👉 A[3]	👉 P[9]	20.832	SLOW
👉 A[3]	👉 P[10]	20.978	SLOW
👉 A[3]	👉 P[11]	22.127	SLOW
👉 A[3]	👉 P[12]	23.093	SLOW
👉 A[3]	👉 P[13]	23.520	SLOW
👉 A[3]	👉 P[14]	23.922	SLOW
👉 A[3]	👉 P[15]	24.467	SLOW
👉 A[4]	👉 P[4]	13.268	SLOW
👉 A[4]	👉 P[5]	14.402	SLOW
👉 A[4]	👉 P[6]	15.935	SLOW
👉 A[4]	👉 P[7]	16.179	SLOW
👉 A[4]	👉 P[8]	17.684	SLOW
👉 A[4]	👉 P[9]	19.843	SLOW
👉 A[4]	👉 P[10]	19.989	SLOW
👉 A[4]	👉 P[11]	21.138	SLOW

Figure 43

➤ A[4]	☑ P[12]	22.104	SLOW
➤ A[4]	☑ P[13]	22.531	SLOW
➤ A[4]	☑ P[14]	22.933	SLOW
➤ A[4]	☑ P[15]	23.478	SLOW
➤ A[5]	☑ P[5]	13.113	SLOW
➤ A[5]	☑ P[6]	15.553	SLOW
➤ A[5]	☑ P[7]	15.494	SLOW
➤ A[5]	☑ P[8]	17.302	SLOW
➤ A[5]	☑ P[9]	19.461	SLOW
➤ A[5]	☑ P[10]	19.608	SLOW
➤ A[5]	☑ P[11]	20.757	SLOW
➤ A[5]	☑ P[12]	21.723	SLOW
➤ A[5]	☑ P[13]	22.150	SLOW
➤ A[5]	☑ P[14]	22.551	SLOW
➤ A[5]	☑ P[15]	23.097	SLOW
➤ A[6]	☑ P[6]	15.263	SLOW
➤ A[6]	☑ P[7]	15.204	SLOW
➤ A[6]	☑ P[8]	17.012	SLOW
➤ A[6]	☑ P[9]	19.171	SLOW
➤ A[6]	☑ P[10]	19.317	SLOW
➤ A[6]	☑ P[11]	20.466	SLOW
➤ A[6]	☑ P[12]	21.432	SLOW
➤ A[6]	☑ P[13]	21.859	SLOW
➤ A[6]	☑ P[14]	22.261	SLOW
➤ A[6]	☑ P[15]	22.806	SLOW
➤ A[7]	☑ P[7]	14.291	SLOW
➤ A[7]	☑ P[8]	16.140	SLOW
➤ A[7]	☑ P[9]	18.258	SLOW
➤ A[7]	☑ P[10]	18.404	SLOW
➤ A[7]	☑ P[11]	19.553	SLOW
➤ A[7]	☑ P[12]	20.519	SLOW
➤ A[7]	☑ P[13]	20.946	SLOW
➤ A[7]	☑ P[14]	21.348	SLOW

Figure 44

👉 A[7]	👉 P[15]	21.893	SLOW
👉 B[0]	👉 P[0]	9.629	SLOW
👉 B[0]	👉 P[1]	9.139	SLOW
👉 B[0]	👉 P[2]	9.154	SLOW
👉 B[0]	👉 P[3]	10.088	SLOW
👉 B[0]	👉 P[4]	13.344	SLOW
👉 B[0]	👉 P[5]	13.597	SLOW
👉 B[0]	👉 P[6]	15.817	SLOW
👉 B[0]	👉 P[7]	15.705	SLOW
👉 B[0]	👉 P[8]	17.253	SLOW
👉 B[0]	👉 P[9]	19.389	SLOW
👉 B[0]	👉 P[10]	19.535	SLOW
👉 B[0]	👉 P[11]	20.684	SLOW
👉 B[0]	👉 P[12]	21.651	SLOW
👉 B[0]	👉 P[13]	22.078	SLOW
👉 B[0]	👉 P[14]	22.479	SLOW
👉 B[0]	👉 P[15]	23.024	SLOW
👉 B[1]	👉 P[1]	8.828	SLOW
👉 B[1]	👉 P[2]	8.765	SLOW
👉 B[1]	👉 P[3]	9.953	SLOW
👉 B[1]	👉 P[4]	13.123	SLOW
👉 B[1]	👉 P[5]	13.489	SLOW
👉 B[1]	👉 P[6]	15.596	SLOW
👉 B[1]	👉 P[7]	15.484	SLOW
👉 B[1]	👉 P[8]	17.032	SLOW
👉 B[1]	👉 P[9]	19.168	SLOW
👉 B[1]	👉 P[10]	19.315	SLOW
👉 B[1]	👉 P[11]	20.464	SLOW
👉 B[1]	👉 P[12]	21.430	SLOW
👉 B[1]	👉 P[13]	21.857	SLOW
👉 B[1]	👉 P[14]	22.258	SLOW
👉 B[1]	👉 P[15]	22.804	SLOW
👉 B[2]	👉 P[2]	8.641	SLOW

Figure 45

👉 B[2]	👉 P[3]	10.493	SLOW
👉 B[2]	👉 P[4]	12.204	SLOW
👉 B[2]	👉 P[5]	13.793	SLOW
👉 B[2]	👉 P[6]	15.289	SLOW
👉 B[2]	👉 P[7]	15.570	SLOW
👉 B[2]	👉 P[8]	17.021	SLOW
👉 B[2]	👉 P[9]	18.742	SLOW
👉 B[2]	👉 P[10]	18.888	SLOW
👉 B[2]	👉 P[11]	20.037	SLOW
👉 B[2]	👉 P[12]	21.004	SLOW
👉 B[2]	👉 P[13]	21.431	SLOW
👉 B[2]	👉 P[14]	21.832	SLOW
👉 B[2]	👉 P[15]	22.377	SLOW
👉 B[3]	👉 P[3]	9.035	SLOW
👉 B[3]	👉 P[4]	11.699	SLOW
👉 B[3]	👉 P[5]	12.086	SLOW
👉 B[3]	👉 P[6]	13.884	SLOW
👉 B[3]	👉 P[7]	14.164	SLOW
👉 B[3]	👉 P[8]	15.615	SLOW
👉 B[3]	👉 P[9]	17.457	SLOW
👉 B[3]	👉 P[10]	17.603	SLOW
👉 B[3]	👉 P[11]	18.752	SLOW
👉 B[3]	👉 P[12]	19.718	SLOW
👉 B[3]	👉 P[13]	20.146	SLOW
👉 B[3]	👉 P[14]	20.547	SLOW
👉 B[3]	👉 P[15]	21.092	SLOW
👉 B[4]	👉 P[4]	9.685	SLOW
👉 B[4]	👉 P[5]	11.459	SLOW
👉 B[4]	👉 P[6]	13.612	SLOW
👉 B[4]	👉 P[7]	13.893	SLOW
👉 B[4]	👉 P[8]	15.344	SLOW
👉 B[4]	👉 P[9]	16.525	SLOW
👉 B[4]	👉 P[10]	16.663	SLOW

Figure 46

B[4]	P[11]	17.812	SLOW
B[4]	P[12]	18.778	SLOW
B[4]	P[13]	19.205	SLOW
B[4]	P[14]	19.607	SLOW
B[4]	P[15]	20.152	SLOW
B[5]	P[5]	9.934	SLOW
B[5]	P[6]	12.851	SLOW
B[5]	P[7]	13.131	SLOW
B[5]	P[8]	14.582	SLOW
B[5]	P[9]	15.763	SLOW
B[5]	P[10]	15.901	SLOW
B[5]	P[11]	17.050	SLOW
B[5]	P[12]	18.016	SLOW
B[5]	P[13]	18.443	SLOW
B[5]	P[14]	18.845	SLOW
B[5]	P[15]	19.390	SLOW
B[6]	P[6]	9.777	SLOW
B[6]	P[7]	13.307	SLOW
B[6]	P[8]	15.148	SLOW
B[6]	P[9]	16.740	SLOW
B[6]	P[10]	16.499	SLOW
B[6]	P[11]	17.648	SLOW
B[6]	P[12]	18.614	SLOW
B[6]	P[13]	19.041	SLOW
B[6]	P[14]	19.443	SLOW
B[6]	P[15]	19.988	SLOW
B[7]	P[7]	9.769	SLOW
B[7]	P[8]	11.403	SLOW
B[7]	P[9]	12.564	SLOW
B[7]	P[10]	12.700	SLOW
B[7]	P[11]	13.773	SLOW
B[7]	P[12]	14.808	SLOW

Figure 47

B[7]	P[13]	14.882	SLOW
B[7]	P[14]	14.898	SLOW
B[7]	P[15]	15.462	SLOW

Figure 48

Here is the power summary for the Brauns multiplier:

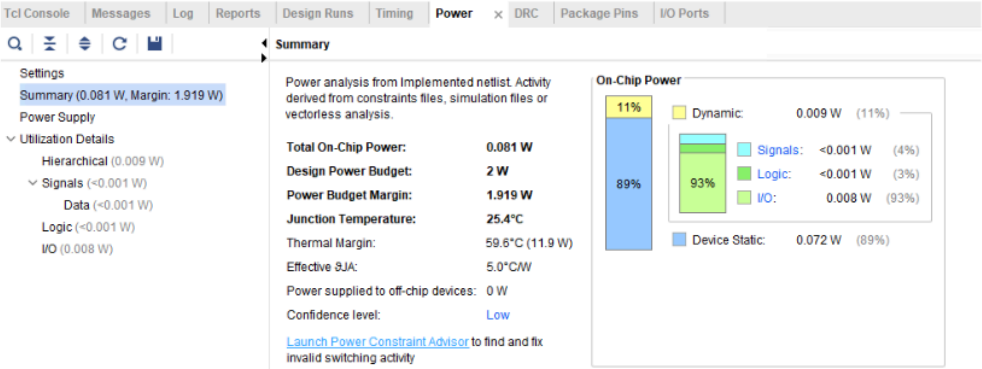


Figure 49

Here is the report utilisation for the Brauns multiplier:

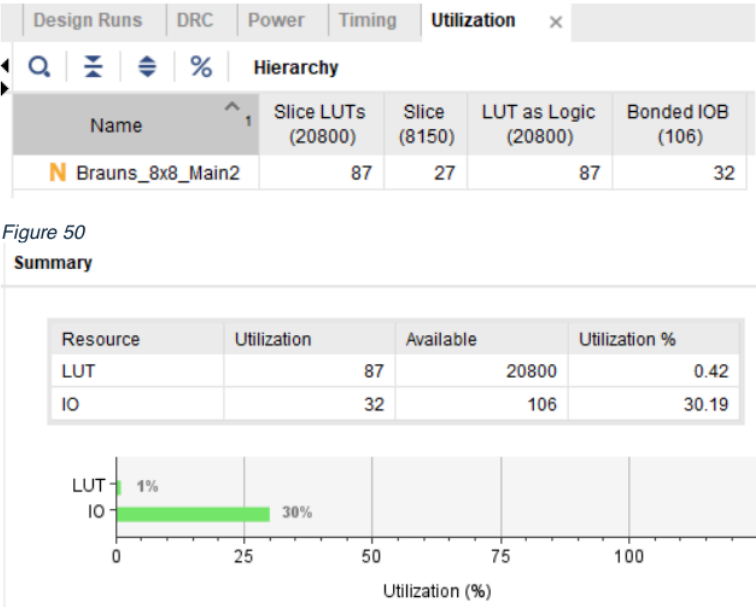


Figure 51

Wallace

Here are the four example calculations inputted into the Wallace Tree multiplier:

EF x FF:

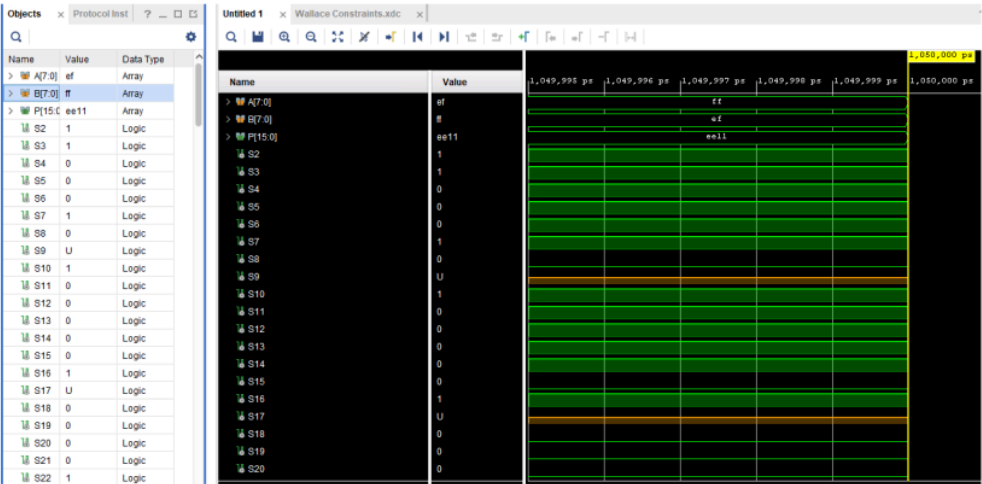


Figure 52

B4 x C9:

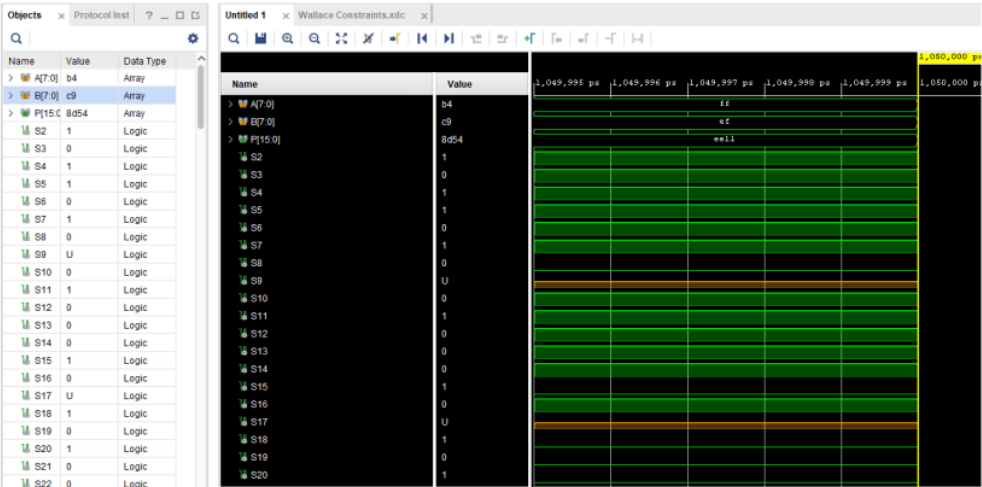


Figure 53

F6 x A1:

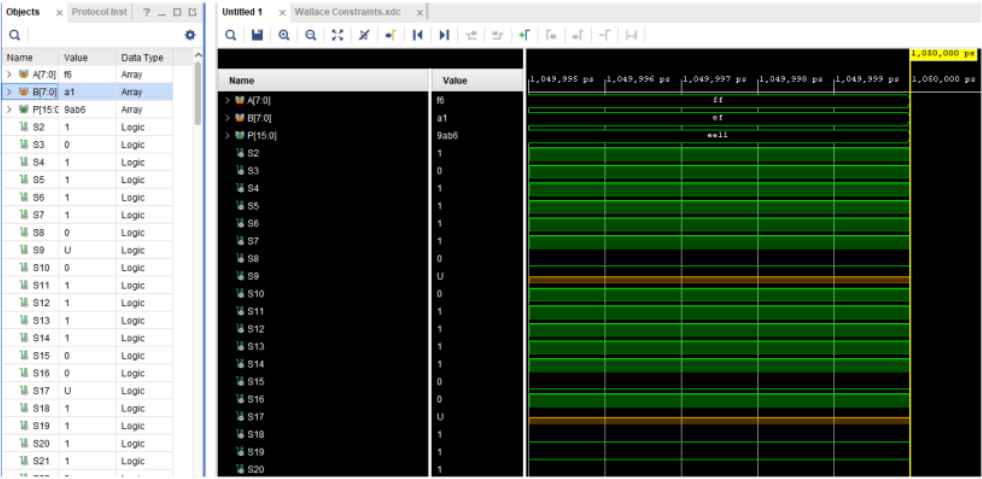


Figure 54

62 x A8:

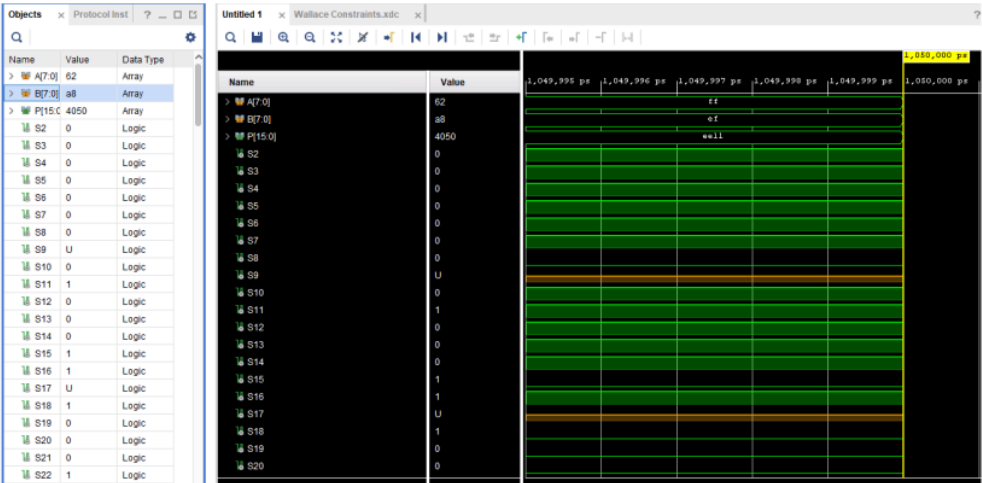


Figure 55

Here is the timing summary for the Wallace Tree multiplier:

Runs

Timing

Power

DRC

Package Pins

I/O Ports

Design Timing Summary

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 6.058 ns	Worst Hold Slack (WHS): NA	Worst Pulse Width Slack (WPWS): NA
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): NA	Total Pulse Width Negative Slack (TPWS): NA
Number of Failing Endpoints: 0	Number of Failing Endpoints: NA	Number of Failing Endpoints: NA
Total Number of Endpoints: 16	Total Number of Endpoints: NA	Total Number of Endpoints: NA

All user specified timing constraints are met.

Figure 56

Here is the combinational delay for the Wallace Tree multiplier:

Combinational Delays				A[2]	P[4]	12.994	SLOW
From Port	To Port	Max Delay	Max Process Corner	A[2]	P[5]	15.016	SLOW
A[0]	P[0]	12.045	SLOW	A[2]	P[6]	14.615	SLOW
A[0]	P[1]	11.480	SLOW	A[2]	P[7]	15.488	SLOW
A[0]	P[2]	10.760	SLOW	A[2]	P[8]	15.458	SLOW
A[0]	P[3]	11.239	SLOW	A[2]	P[9]	16.618	SLOW
A[0]	P[4]	12.166	SLOW	A[2]	P[10]	17.892	SLOW
A[0]	P[5]	14.188	SLOW	A[2]	P[11]	17.268	SLOW
A[0]	P[6]	13.787	SLOW	A[2]	P[12]	18.906	SLOW
A[0]	P[7]	14.660	SLOW	A[2]	P[13]	19.498	SLOW
A[0]	P[8]	14.630	SLOW	A[2]	P[14]	18.999	SLOW
A[0]	P[9]	15.790	SLOW	A[2]	P[15]	19.636	SLOW
A[0]	P[10]	17.064	SLOW	A[3]	P[3]	12.015	SLOW
A[0]	P[11]	16.440	SLOW	A[3]	P[4]	12.937	SLOW
A[0]	P[12]	18.078	SLOW	A[3]	P[5]	14.959	SLOW
A[0]	P[13]	18.669	SLOW	A[3]	P[6]	14.558	SLOW
A[0]	P[14]	18.171	SLOW	A[3]	P[7]	15.431	SLOW
A[0]	P[15]	18.808	SLOW	A[3]	P[8]	15.402	SLOW
A[1]	P[1]	11.809	SLOW	A[3]	P[9]	16.561	SLOW
A[1]	P[2]	11.427	SLOW	A[3]	P[10]	17.836	SLOW
A[1]	P[3]	11.553	SLOW	A[3]	P[11]	17.211	SLOW
A[1]	P[4]	12.530	SLOW	A[3]	P[12]	18.849	SLOW
A[1]	P[5]	14.584	SLOW	A[3]	P[13]	19.441	SLOW
A[1]	P[6]	14.391	SLOW	A[3]	P[14]	18.942	SLOW
A[1]	P[7]	15.056	SLOW	A[3]	P[15]	19.580	SLOW
A[1]	P[8]	15.026	SLOW	A[4]	P[4]	12.071	SLOW
A[1]	P[9]	16.186	SLOW	A[4]	P[5]	14.125	SLOW
A[1]	P[10]	17.460	SLOW	A[4]	P[6]	14.069	SLOW
A[1]	P[11]	16.836	SLOW	A[4]	P[7]	14.724	SLOW
A[1]	P[12]	18.474	SLOW	A[4]	P[8]	14.909	SLOW
A[1]	P[13]	19.005	SLOW	A[4]	P[9]	16.207	SLOW
A[1]	P[14]	18.567	SLOW	A[4]	P[10]	17.202	SLOW
A[1]	P[15]	19.204	SLOW	A[4]	P[11]	16.915	SLOW
A[2]	P[2]	10.944	SLOW				
A[2]	P[3]	12.071	SLOW				

Figure 57

A[4]	P[12]	18.741	SLOW	A[7]	P[15]	19.917	SLOW
A[4]	P[13]	19.333	SLOW	B[0]	P[0]	9.791	SLOW
A[4]	P[14]	18.834	SLOW	B[0]	P[1]	9.229	SLOW
A[4]	P[15]	19.471	SLOW	B[0]	P[2]	8.497	SLOW
A[5]	P[5]	13.763	SLOW	B[0]	P[3]	10.149	SLOW
A[5]	P[6]	14.085	SLOW	B[0]	P[4]	11.072	SLOW
A[5]	P[7]	14.679	SLOW	B[0]	P[5]	13.094	SLOW
A[5]	P[8]	14.925	SLOW	B[0]	P[6]	12.926	SLOW
A[5]	P[9]	16.223	SLOW	B[0]	P[7]	13.566	SLOW
A[5]	P[10]	17.218	SLOW	B[0]	P[8]	13.947	SLOW
A[5]	P[11]	16.621	SLOW	B[0]	P[9]	15.256	SLOW
A[5]	P[12]	18.447	SLOW	B[0]	P[10]	16.252	SLOW
A[5]	P[13]	19.039	SLOW	B[0]	P[11]	15.813	SLOW
A[5]	P[14]	18.541	SLOW	B[0]	P[12]	17.639	SLOW
A[5]	P[15]	19.178	SLOW	B[0]	P[13]	18.231	SLOW
A[6]	P[6]	14.019	SLOW	B[0]	P[14]	17.733	SLOW
A[6]	P[7]	14.614	SLOW	B[0]	P[15]	18.370	SLOW
A[6]	P[8]	15.145	SLOW	B[1]	P[1]	9.507	SLOW
A[6]	P[9]	16.455	SLOW	B[1]	P[2]	8.890	SLOW
A[6]	P[10]	17.450	SLOW	B[1]	P[3]	10.158	SLOW
A[6]	P[11]	17.115	SLOW	B[1]	P[4]	11.487	SLOW
A[6]	P[12]	18.941	SLOW	B[1]	P[5]	13.541	SLOW
A[6]	P[13]	19.533	SLOW	B[1]	P[6]	13.382	SLOW
A[6]	P[14]	19.035	SLOW	B[1]	P[7]	14.013	SLOW
A[6]	P[15]	19.672	SLOW	B[1]	P[8]	14.281	SLOW
A[7]	P[7]	14.426	SLOW	B[1]	P[9]	15.590	SLOW
A[7]	P[8]	15.494	SLOW	B[1]	P[10]	16.586	SLOW
A[7]	P[9]	16.803	SLOW	B[1]	P[11]	16.147	SLOW
A[7]	P[10]	17.799	SLOW	B[1]	P[12]	17.973	SLOW
A[7]	P[11]	17.360	SLOW	B[1]	P[13]	18.565	SLOW
A[7]	P[12]	19.186	SLOW	B[1]	P[14]	18.067	SLOW
A[7]	P[13]	19.778	SLOW	B[1]	P[15]	18.704	SLOW
A[7]	P[14]	19.280	SLOW	B[2]	P[2]	8.765	SLOW

Figure 58

B[2]	P[3]	10.398	SLOW	B[4]	P[11]	15.144	SLOW
B[2]	P[4]	11.664	SLOW	B[4]	P[12]	16.782	SLOW
B[2]	P[5]	13.717	SLOW	B[4]	P[13]	17.374	SLOW
B[2]	P[6]	13.525	SLOW	B[4]	P[14]	16.876	SLOW
B[2]	P[7]	14.190	SLOW	B[4]	P[15]	17.513	SLOW
B[2]	P[8]	14.373	SLOW	B[5]	P[5]	10.141	SLOW
B[2]	P[9]	15.682	SLOW	B[5]	P[6]	12.342	SLOW
B[2]	P[10]	16.678	SLOW	B[5]	P[7]	12.674	SLOW
B[2]	P[11]	16.353	SLOW	B[5]	P[8]	13.380	SLOW
B[2]	P[12]	18.179	SLOW	B[5]	P[9]	14.689	SLOW
B[2]	P[13]	18.770	SLOW	B[5]	P[10]	15.685	SLOW
B[2]	P[14]	18.272	SLOW	B[5]	P[11]	15.056	SLOW
B[2]	P[15]	18.909	SLOW	B[5]	P[12]	16.694	SLOW
B[3]	P[3]	9.109	SLOW	B[5]	P[13]	17.286	SLOW
B[3]	P[4]	10.882	SLOW	B[5]	P[14]	16.787	SLOW
B[3]	P[5]	12.936	SLOW	B[5]	P[15]	17.425	SLOW
B[3]	P[6]	12.773	SLOW	B[6]	P[6]	10.429	SLOW
B[3]	P[7]	13.494	SLOW	B[6]	P[7]	12.239	SLOW
B[3]	P[8]	13.597	SLOW	B[6]	P[8]	12.336	SLOW
B[3]	P[9]	14.906	SLOW	B[6]	P[9]	13.613	SLOW
B[3]	P[10]	16.002	SLOW	B[6]	P[10]	14.665	SLOW
B[3]	P[11]	15.517	SLOW	B[6]	P[11]	14.041	SLOW
B[3]	P[12]	17.344	SLOW	B[6]	P[12]	15.679	SLOW
B[3]	P[13]	17.935	SLOW	B[6]	P[13]	16.270	SLOW
B[3]	P[14]	17.437	SLOW	B[6]	P[14]	15.772	SLOW
B[3]	P[15]	18.074	SLOW	B[6]	P[15]	16.409	SLOW
B[4]	P[4]	9.641	SLOW	B[7]	P[7]	9.516	SLOW
B[4]	P[5]	11.697	SLOW	B[7]	P[8]	10.815	SLOW
B[4]	P[6]	12.535	SLOW	B[7]	P[9]	12.583	SLOW
B[4]	P[7]	13.211	SLOW	B[7]	P[10]	13.858	SLOW
B[4]	P[8]	13.468	SLOW	B[7]	P[11]	13.310	SLOW
B[4]	P[9]	14.777	SLOW	B[7]	P[12]	14.948	SLOW
B[4]	P[10]	15.773	SLOW	B[7]	P[13]	15.539	SLOW

Figure 59

B[7]	P[14]	15.041	SLOW
B[7]	P[15]	15.678	SLOW

Figure 60

Here is the power summary for the Wallace Tree multiplier:

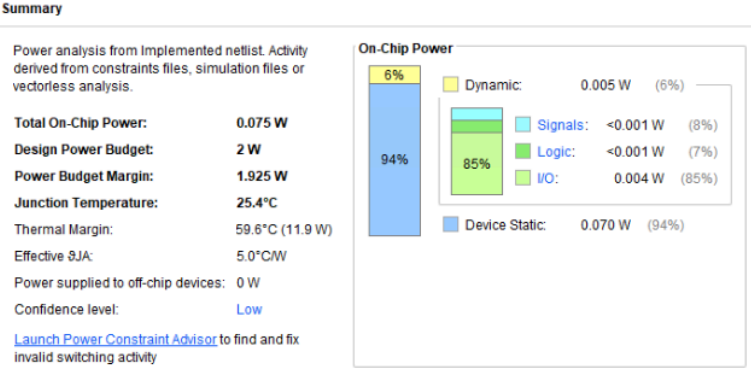


Figure 61

Here is the report utilisation for the Wallace Tree multiplier:

Design Runs	Utilization	Timing	Power	DRC	Package Pins	I/O
Hierarchy						
Name	^1	Slice LUTs (20800)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	
N Wallace_8x8_Main		81	23	81	32	

Figure 62

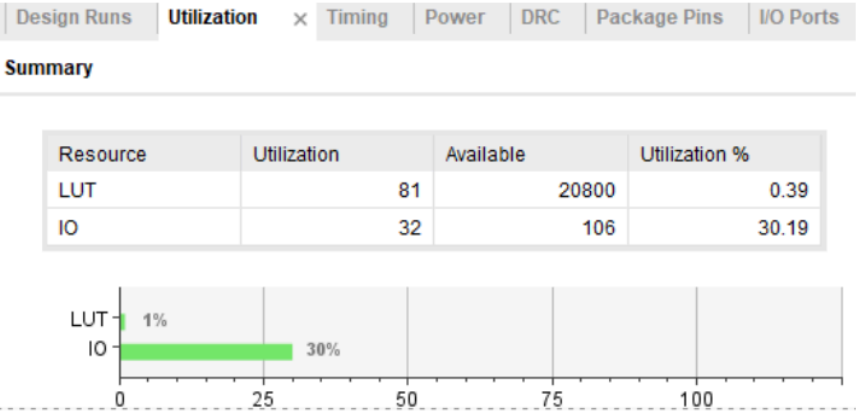


Figure 63

Analysis of Results

As shown above in Figures 37-41 and 52-55, both the Brauns and Wallace Tree multipliers were tested for correct operation. These were tested using the hexadecimal values:

1. EF x FF = EE11
2. B4 x C9 = 8D54
3. F6 x A1 = 9AB6
4. 62 x A8 = 4050

Both multipliers can be seen providing the correct results for these multiplications, meaning they both function as intended.

The critical path time calculated above was lower for the Wallace Tree multiplier in comparison to the Brauns multiplier. Due to this, it would be expected that the Wallace Tree multiplier would conduct its operation quicker than the Brauns multiplier. By looking at the timing summaries for both multipliers above in Figures 41 and 56, it can be observed that the worst negative slack (WNS) for the Brauns multiplier was 13.508ns whereas the WNS for the Wallace Tree multiplier was 6.058ns. WNS is the time left of a clock cycle after the multiplier has conducted an operation along its critical path [14]. As the clock cycle for the Brauns multiplier was 41ns and 29ns for the Wallace Tree multiplier, it took them 27.492ns and 22.948ns to conduct an operation along its critical path successfully, respectively. This evidence shows that the Wallace Tree multiplier is faster than a Brauns multiplier since a Brauns multiplier has more hardware along its critical path compared to a Wallace Tree multiplier.

The combinational delays shown in Figures 42-48 and 57-60 also prove that the Wallace Tree multiplier is faster than a Brauns multiplier at conducting an operation along its critical path successfully. For each port A input (A0-A7) for P15, there is a greater maximum delay for the Brauns multiplier compared to the Wallace Tree multiplier. An example of this is that A3-P15 has a delay of 24.467ns for the Brauns multiplier whereas the Wallace Tree multiplier only has a delay of 19.580ns. This displays further proof that the Wallace Tree multiplier is faster than a Brauns multiplier. The same can be said for each port B input (B0-B7) for P15. Another example of this is that B6-P15 has a delay of 19.998ns for the Brauns multiplier whereas the Wallace Tree multiplier only has a delay of 16.409ns. This is most likely due to the Brauns multipliers overreliance of Ripple-Carry adders and the Wallace Trees utilisation of its pathing using Carry-Sum adders.

Also shown in Figures 49 and 61 are the power summaries for both multipliers. The Brauns multiplier has a total on-chip power usage of 0.081W. The Wallace Tree multiplier has a total on-chip power usage of 0.075W. Both multipliers had the same design power budget of 2W. With this information it can be deduced that the Wallace Tree multiplier is also more efficient in its power usage than the Brauns multiplier. This again occurs because the Wallace Tree multiplier is more efficient in its hardware usage compared to the Brauns multiplier but is also more structurally efficient. Again, due to there being a lower number of Ripple-Carry adders within the Wallace Tree multiplier compared to the Brauns multiplier, less full adders are being used to conduct a single operation, making the Wallace Tree multiplier more structurally and power efficient.

Finally, the report utilisations above in Figures 50-51 and 62-63 show the number of used and available LUTs (look-up tables) for both multipliers. Both multipliers have 20800 LUTs and 106 I/Os available to them. Although they both use the same amount of I/Os (can also be seen in the pinouts above), the Brauns multiplier uses 87 (or 0.42%) of its LUTs whereas the Wallace Tree multiplier uses 81 (or 0.39%) of its LUTs. This demonstrates that even if it's by a small amount, the Wallace Tree multiplier is more efficient in hardware usage compared to the Brauns multiplier. In addition to this, it also shows that spatially the Wallace Tree multiplier is more efficient when implemented in an FPGA.  11

Testing Details & Recommendations for Future Improvements

Recommendations for future improvements are as follows:

- Potentially scale multipliers up to 32 bit
- Analyse combinational delays in further detail by calculating a mean time difference between the two multipliers
- Conduct further research into the implementation of Brauns and Wallace Tree multipliers  12
- Conduct further research into different adders and multipliers

Conclusion

It can be concluded that the Wallace Tree multiplier is faster in operational speed, power and space efficiency when compared to that of the Brauns multiplier. This is due to the Wallace Tree multiplier's structure and use of adders, as well as its utilisation of partial products. It does not over rely on adders chained together like the Brauns multiplier and sends its partial product outputs throughout its tree. The evidence gained from testing has proven this throughout the report.

References

- [C. A. MADHU GUDHIMETLA, "COMPARISON OF DIFFERENT TYPES OF 1 MULTIPLIERS WITH," in *IRF International Conference*, Bengaluru, India, 2017.]
- [W. Green, "Multiplication with FPGA DSPs," Project F, 7 3 2023. [Online]. 2 Available: <https://projectf.io/posts/multiplication-fpga-dsps/>. [Accessed 15 4 2025].]

- [B. S. N. S. K. N. D. Prasad, "Review on Different Types of Multipliers and Its
3 Performance Comparisons," in *Advances in Smart Grid and Renewable Energy*,
] Springer, Singapore, 2021.
- [Interview Cake, "Bit Shifting," Interview Cake, [Online]. Available:
4 [https://www.interviewcake.com/concept/java/bit-](https://www.interviewcake.com/concept/java/bit-shift#:~:text=Logical%20Right%20Shifts,inserted%20on%20the%20other%20end.&text=For%20positive%20numbers%2C%20a%20single,2%2C%20throwing%20out%20any%20remainders..)
] [shift#:~:text=Logical%20Right%20Shifts,inserted%20on%20the%20other%20end
&text=For%20positive%20numbers%2C%20a%20single,2%2C%20throwing%20
out%20any%20remainders..](https://www.interviewcake.com/concept/java/bit-shift#:~:text=Logical%20Right%20Shifts,inserted%20on%20the%20other%20end.&text=For%20positive%20numbers%2C%20a%20single,2%2C%20throwing%20out%20any%20remainders..) [Accessed 15 4 2025].
- [Unknown, "Structure of Computer Systems," Unknown Unknown Unknown.
5 [Online]. Available: [https://users.utcluj.ro/~baruch/book_ssce/SSCE-Shift-](https://users.utcluj.ro/~baruch/book_ssce/SSCE-Shift-Mult.pdf)
] [Mult.pdf](https://users.utcluj.ro/~baruch/book_ssce/SSCE-Shift-Mult.pdf). [Accessed 15 4 2025].
- [R. Andraka, "Multiplication," Andraka Consulting Group, Inc, Unknown Unknown
6 Unknown. [Online]. Available: <http://www.andraka.com/multipli.php>. [Accessed 15
] 4 2025].
- [R. S. K. S. S. a. R. S. R. P. Kishore, "Implementation of Braun and Baugh-Wooley
7 Multipliers Using QCA," in *2nd International Conference for Innovation in
] Technology (INOCON)*, Bangalore, India, 2023.
- [A. Sudhakar, "Digital Design — Ripple Carry Adders," Medium.com, 1 7 2024.
8 [Online]. Available: [https://medium.com/@abhinavsudhakar14/digital-design-](https://medium.com/@abhinavsudhakar14/digital-design-ripple-carry-adders-486305f69b23)
] [ripple-carry-adders-486305f69b23](https://medium.com/@abhinavsudhakar14/digital-design-ripple-carry-adders-486305f69b23). [Accessed 15 4 2025].
- [NxtWave, "Booth's Multiplication Algorithm in Computer Organization," NxtWave,
9 29 11 2024. [Online]. Available: [https://www.ccbp.in/blog/articles/booth-](https://www.ccbp.in/blog/articles/booth-multiplication-algorithm-in-computer-organization)
] [multiplication-algorithm-in-computer-organization](https://www.ccbp.in/blog/articles/booth-multiplication-algorithm-in-computer-organization). [Accessed 15 4 2025].
- [T. Finley, "Two's Complement," 4 2000. [Online]. Available:
1 [https://www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html#:~:text=Two%27s
0 %20complement%20is%20the%20way,add%20one%20to%20the%20result..](https://www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html#:~:text=Two%27s%20complement%20is%20the%20way,add%20one%20to%20the%20result..)
] [Accessed 15 4 2025].
- [V. S. a. H. M. K. M. I. B. Ramkumar, "A Design Technique for Faster Dadda
1 Multiplier".
1
]
- [D. S. ROY, "Wallace and Dadda Multiplier Design," 18 6 2019. [Online]. Available:
1 <https://digitalsystemdesign.in/wallace-and-dadda-multiplier-design/>. [Accessed 15
2 4 2025].
]

[C. BasuMallick, "What Is FPGA (Field-Programmable Gate Array)? Meaning,
1 Working, and Uses," 17 10 2022. [Online]. Available:
3 <https://www.spiceworks.com/tech/networking/articles/what-is-fpga/>. [Accessed 15
] 4 2025].

[WordPress.com, "Determining the maximum operating frequency in Xilinx
1 Vivado," 8 8 2019. [Online]. Available:
4 <https://verilogbeginner.wordpress.com/tag/worst-negative-slack/>. [Accessed 15 4
] 2025].

Electronics 2 Assignment - Ruben Bhatia - 2104693

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

63/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

 **Comment 1** |  Engagement with literature(20)

The basic idea of binary multiplication is seen.

 **Comment 2** |  Engagement with literature(20)

The discussion of various multiplier types is a positive aspect.

PAGE 5

 **Comment 3** |  Knowledge & Understanding(20)

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

 **Comment 4** |  Engagement with literature(20)

The inclusion of their applications and advantages provides useful context.

Comment 5 | Knowledge & Understanding(20)

The longest delay prediction diagram is missing. It would be better to show manual delay calculations step-by-step to prove understanding.

Comment 6 | Knowledge & Understanding(20)

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



Comment 7 | Practical Application (25)

screenshots of XDC files shows the actual pin assignments used for implementation.



Comment 8 | Practical Application (25)

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



Comment 9 | Practical Application (25)

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



Comment 10 | Cognitive skills (20)

The report includes s, power analysis screenshots, LUT usage data, and area details, which shows effort towards evaluating the design performance and resource utilisation.

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

PAGE 50

PAGE 51



Comment 11 | Cognitive skills (20)

The report presents output performance analysis using parameters such as power, LUT usage, speed, and area, which indicates that the designs were evaluated thoroughly.



Comment 12 | Cognitive skills (20)

The future recommendations are well noted and show a good understanding of how the work can be taken forward.



Comment 13 | Transferable skills (15)

The report is arranged nicely with clear sections.

PAGE 52

PAGE 53

GRADING FORM: UNTITLED

ENGAGEMENT WITH LITERATURE(20)

Assesses clarity of definition, technical explanation, advantages, disadvantages, applications, and use of relevant references. Also includes critical analysis, research gaps, and proper citation and referencing.



Basic use of literature with limited relevance or depth. References may be likely related to the discussion. Critical analysis is minimal . Gaps in research are limited. Citation style is inconsistent, and source variety is limited.

11.8

KNOWLEDGE & UNDERSTANDING(20)

This section evaluates understanding of binary multiplication, multiplier architectures, logic design, and adders. It also includes key concepts like power efficiency, delay path, LUTs, and slack, reflecting application of core engineering principles.



Adequate but limited understanding. Key topics are mentioned (e.g., Braun, Wallace, logic design), but with shallow explanation. Neccessity for the aspects like delay paths, power basics, and slack needs more explanation.

10.2

COGNITIVE SKILLS (20)

This section includes analysis of design structures, performance interpretation, and comparison of results using appropriate techniques. Emphasis is placed on understanding technical relationships, interpreting data, and drawing meaningful conclusions with supporting evidence such as calculations, graphs, and recommendation

9.7



Shows strong analytical ability. Most design analyses are accurate, with logical interpretation of delay and performance parameters. However, the dot diagram, which is essential for understanding the structure of the multiplier, is missing. Additionally, the longest delay path diagram and the corresponding delay calculations have not been provided, which are important for evaluating the design's timing performance.

PRACTICAL APPLICATION (25)

This section assesses the practical skills in VHDL implementation, port mapping, FPGA pin assignment, simulation, and hardware proof. It also includes synthesis, applying XDC constraints, and interpreting reports on LUTs, power, delay, and timing, reflecting effective tool usage.

20.5



Very good execution of practical tasks. VHDL code is functionally sound; Pin assignments are mostly correct. Simulation and hardware outputs are included with minor gaps. FPGA synthesis and report handling are well managed. Constraints are applied with general accuracy. The report also lacks a clear explanation of the port mapping considerations

TRANSFERABLE SKILLS (15)

This section evaluates the ability to communicate work clearly and effectively. It covers report writing, clarity of diagrams, structured presentation, and appropriate use of screenshots. It also assesses how well findings are explained, the accuracy with which procedures are followed, and how results are summarised. Communication should be suited to both technical and non-technical audiences, demonstrating professionalism and clarity.



Very good communication. Report is coherent and includes clear diagrams and screenshots. Findings are explained well and structure is mostly consistent. Procedural accuracy and summary are evident. Minor improvements could be made in tailoring language for diverse audiences.

11