

DSS

by Daniel Lennon

FILE	2792_DANIEL_LENNON_DSS_114255_779322043.DOCX		
TIME SUBMITTED	11-MAY-2018 03:01PM (UTC+0100)	WORD COUNT	1482
SUBMISSION ID	88999541	CHARACTER COUNT	7722



PRIFYSGOL CYMRU

Y Drindod Dewi Sant

UNIVERSITY OF WALES

Trinity Saint David

SWANSEA - ABERTAWE

Design of an 8 x 8 signed number
multiplier

By Daniel Lennon.

Digital System synthesis.

Tutor Dr. Gary Jones.

Level 6

Contents

1. Introduction.	3
2. introduction to VHDL.	4
3. Half and full adders.	4
4. 8 x 8 multiplier design.	5
5. conclusion	22
6. references	22

1. Introduction.

The purpose of this assignment was to design, simulate and synthesise an 8 bit x 8 bit signed number multiplier based upon a simple parallel Braun multiplier. This was done using the Xilinx ISE design software and the program was written using the VHDL hardware description language.

Multiplication when using binary numbers is carried out in much the same way as when multiplying decimal numbers, by calculating partial products, shifting them to the left and then adding them together. When doing this with binary numbers as there are only two possible results, '1' or '0' it is much simpler and just involves shifting and adding and not any large number multiplications. An 8 x 8 multiplier takes two 8-bit binary numbers and multiplies them to create a 16-bit result. This is done by using a combination of AND gates, half and full adders.

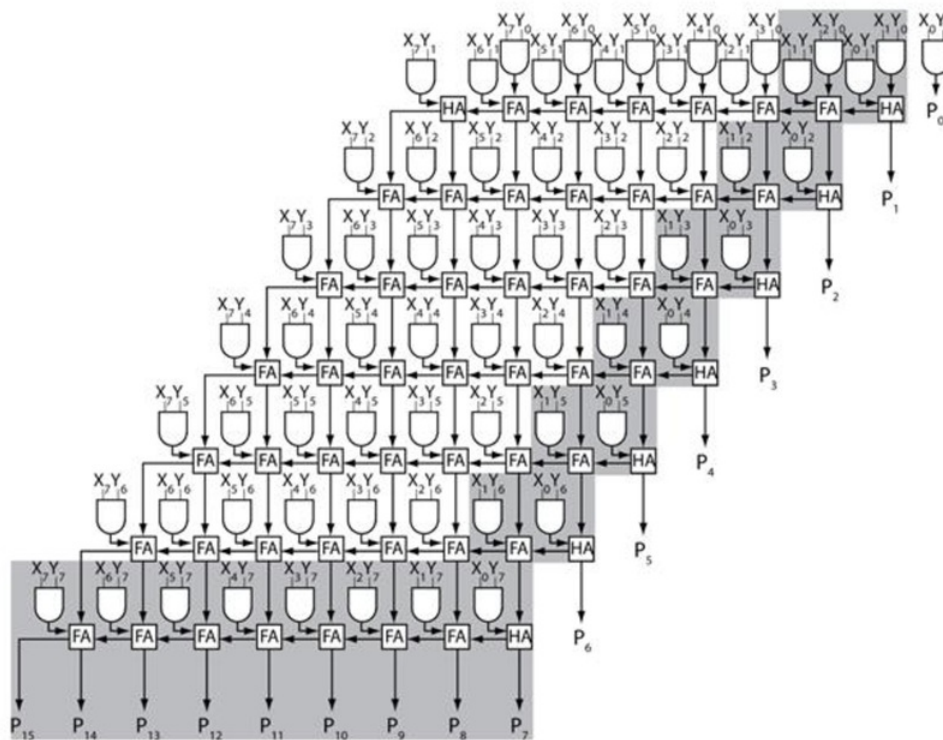


Figure 1. example 8 x 8 multiplier.

Above is an example of the structure of a parallel 8 x 8 Braun multiplier, also known as a carry save array multiplier[1] showing the use of AND gates and half and full adders.

2. introduction to VHDL.

VHDL (Very high speed integrated circuit Hardware Description Language) is one of two languages used to design FPGAs (Field Programmable Gate Arrays) and ASICs (Application Specific Integrated Circuits). The difference between FPGAs and ASICs is that an FPGA is reprogrammable where as an ASIC is not. An FPGA is a component that can be made up of many gates and flip flops and can be used to perform multiple operations in parallel with each other. They are used in many high-speed, high-performance tasks such as image processing, telecommunications, digital signal processing and high-frequency stock market trading[2]. VHDL was developed in the mid-1980's by the U.S. Department of Defence and the IEEE, with the aim of creating very high-speed integrated circuits. A hardware description language is inherently parallel, meaning that the commands are executed in parallel as soon as a new input arrives. A HDL program mimics the behaviour of a physical digital system and it also allows incorporation of timing specifications as well as to describe a system as an interconnection of different components[3].

3. Half and full adders.

Adders are digital circuits that can add binary numbers. A half adder can take two binary numbers and produce a sum of the two plus a carry value. A Half Adder is usually constructed from an AND gate and XOR gate.

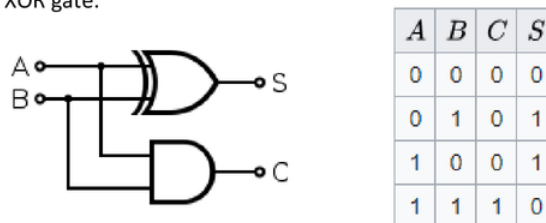


Figure 2. Half adder design and truth table.

A full adder adds three inputs to create two outputs. Like the half adder it can add two signals A and B but also includes a third input C_{in} which is the carry in product of a previous adder. This type of adder can be combined with others to create a Byte wide adder cascading the carry bit from one adder to another. This type of adder can be designed with logic gates by combining two half adders.

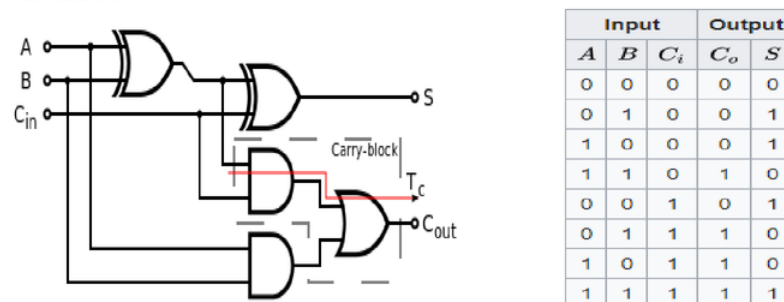


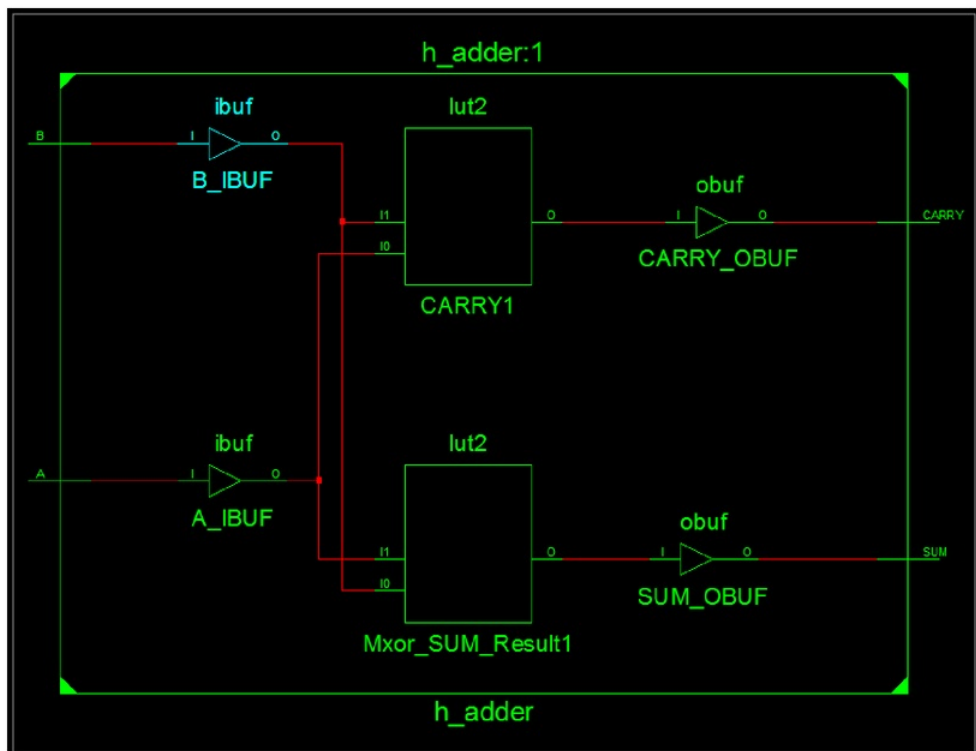
Figure 3. full adder design and truth table.

4. 8 x 8 multiplier design.

To design the 8x8 multiplier for this assignment several components needed to be created, the full adder, the half adder, line 1 and line x. These components were then combined to create the full 8 x 8 multiplier. The following diagrams show the code written to create each of the individual components used in this design.

```
1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    09:44:13 03/08/2018
6  -- Design Name:
7  -- Module Name:    h_adder - 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 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity h_adder is
33     Port ( A : in  STD_LOGIC;
34           B : in  STD_LOGIC;
35           SUM : out STD_LOGIC;
36           CARRY : out STD_LOGIC);
37 end h_adder;
38
39 architecture Behavioral of h_adder is
40
41 begin
42     SUM <= A XOR B;
43     CARRY <= a AND B;
44
45 end Behavioral;
46
47
```

The above diagram shows the code written to create the half adder. Lines 32 to 36 define the inputs A and B and the outputs SUM and CARRY of the adder and lines 42 and 43 determine the actions to be carried out using the logic functions XOR and AND.



The diagram above shows the schematic created after the half adder had been designed.

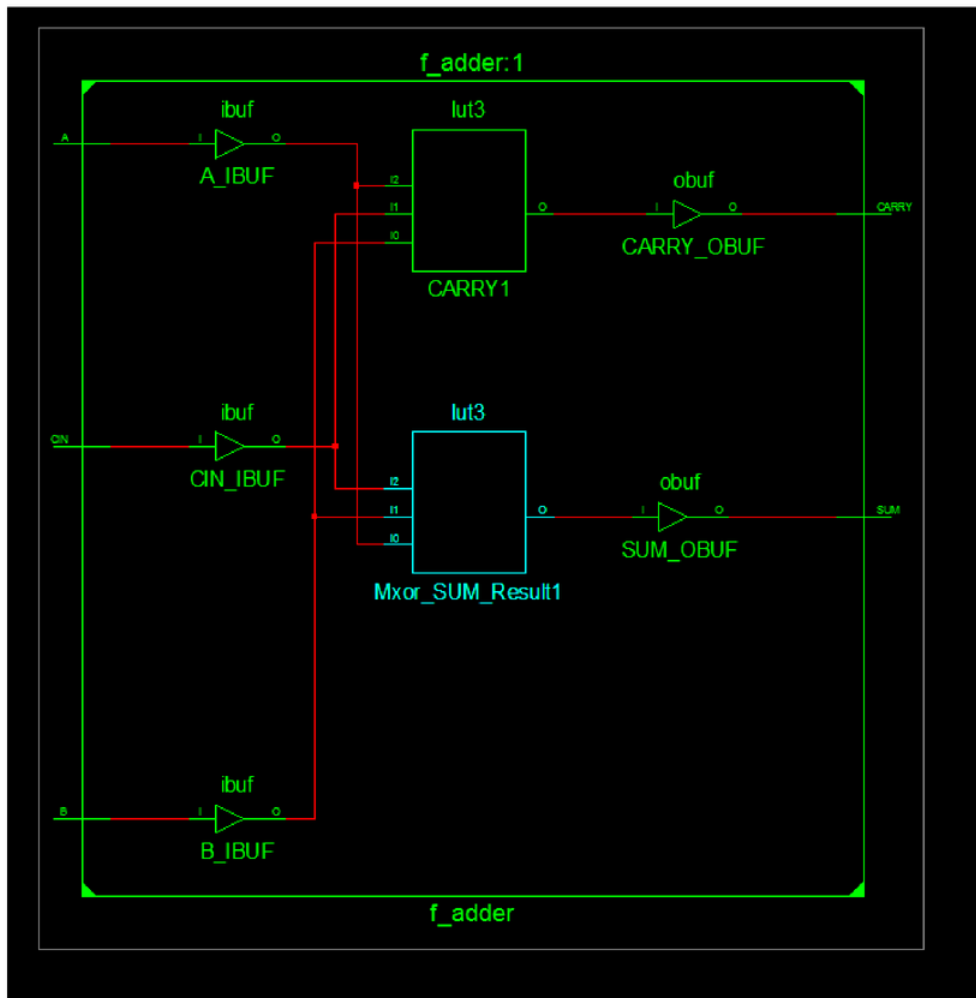
```

19
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity f_adder is
33     Port ( A : in  STD_LOGIC;
34           B : in  STD_LOGIC;
35           CIN : in  STD_LOGIC;
36           SUM : out STD_LOGIC;
37           CARRY : out STD_LOGIC);
38 end f_adder;
39
40 architecture Behavioral of f_adder is
41
42 begin
43     SUM <= A XOR B XOR CIN;
44     CARRY <= (A AND B ) OR ((A XOR B) AND CIN);
45
46 end Behavioral;

```

The diagram above shows the code written to create the full adder. Lines 32 to 38 define the inputs A, B and CIN and the outputs SUM and CARRY. Line 43 determines the process of creating the

output SUM from the inputs A, B and CIN using XOR gates and line 44 shows the process used to create the CARRY output using AND, OR and XOR gate logic.



Above is the schematic created after the design of the full adder showing the addition of the input CIN to the component.

After designing the full and half adder the next step in the design was to develop line 1 of the multiplier. This line takes the 8 bit inputs A0 to A7 and adds them to the input B using simple AND gate logic.

```

31
32 entity L1_8X8 is                                -- SET INPUTS/OUTPUTS FOR LINE 1
33     Port ( A0 : in  STD_LOGIC;
34            A1 : in  STD_LOGIC;
35            A2 : in  STD_LOGIC;
36            A3 : in  STD_LOGIC;
37            A4 : in  STD_LOGIC;
38            A5 : in  STD_LOGIC;
39            A6 : in  STD_LOGIC;
40            A7 : in  STD_LOGIC;
41            B  : in  STD_LOGIC;
42            Q0 : out STD_LOGIC;
43            Q1 : out STD_LOGIC;
44            Q2 : out STD_LOGIC;
45            Q3 : out STD_LOGIC;
46            Q4 : out STD_LOGIC;
47            Q5 : out STD_LOGIC;
48            Q6 : out STD_LOGIC;
49            Q7 : out STD_LOGIC;
50     );
51 end L1_8X8;
52
53 architecture Behavioral of L1_8X8 is
54
55 begin
56     -- Q0-7 BECOME THE RESULTS OF A0-7 ANDED WITH B
57     Q0 <= A0 AND B;
58     Q1 <= A1 AND B;
59     Q2 <= A2 AND B;
60     Q3 <= A3 AND B;
61     Q4 <= A4 AND B;
62     Q5 <= A5 AND B;
63     Q6 <= A6 AND B;
64     Q7 <= A7 AND B;
65
66 end Behavioral;
67

```

In this section of the code lines 33 to 49 define the inputs A0 to A7, B and the outputs Q0 to Q7. Lines 57 to 64 perform the logic functions of ANDing the A inputs with the B input to create the outputs Q0 to Q7.

Once this line had been created a test bench was developed to test that the code functioned in the correct way.

```
38 ARCHITECTURE behavior OF LINE1_8_8_TB IS
39
40     -- Component Declaration for the Unit Under Test (UUT)
41
42     COMPONENT L1_8X8
43     PORT (
44         A0 : IN  std_logic;  --SET PORTS A AND B AS INPUTS AND Q AS OUTPUTS
45         A1 : IN  std_logic;
46         A2 : IN  std_logic;
47         A3 : IN  std_logic;
48         A4 : IN  std_logic;
49         A5 : IN  std_logic;
50         A6 : IN  std_logic;
51         A7 : IN  std_logic;
52         B  : IN  std_logic;
53         Q0 : OUT std_logic;
54         Q1 : OUT std_logic;
55         Q2 : OUT std_logic;
56         Q3 : OUT std_logic;
57         Q4 : OUT std_logic;
58         Q5 : OUT std_logic;
59         Q6 : OUT std_logic;
60         Q7 : OUT std_logic;
61     );
62     END COMPONENT;
63
64     --Inputs
65     signal A0 : std_logic := '0';  -- SET ALL INPUTS TO 0
66     signal A1 : std_logic := '0';
67     signal A2 : std_logic := '0';
68     signal A3 : std_logic := '0';
69     signal A4 : std_logic := '0';
70     signal A5 : std_logic := '0';
71     signal A6 : std_logic := '0';
72     signal A7 : std_logic := '0';
73     signal B  : std_logic := '0';
74
75     --Outputs
76     signal Q0 : std_logic;  -- SET OUTPUTS
77     signal Q1 : std_logic;
78     signal Q2 : std_logic;
79     signal Q3 : std_logic;
80     signal Q4 : std_logic;
81     signal Q5 : std_logic;
82     signal Q6 : std_logic;
83     signal Q7 : std_logic;
```

```

90  -- Instantiate the Unit Under Test (UUT)
91  uut: L1_SX8 PORT MAP (
92      A0 => A0,    -- ASSIGNE VALUES TO THEMSELVES
93      A1 => A1,
94      A2 => A2,
95      A3 => A3,
96      A4 => A4,
97      A5 => A5,
98      A6 => A6,
99      A7 => A7,
100     B => B,
101     Q0 => Q0,
102     Q1 => Q1,
103     Q2 => Q2,
104     Q3 => Q3,
105     Q4 => Q4,
106     Q5 => Q5,
107     Q6 => Q6,
108     Q7 => Q7
109 );
110
111
112
113
114  -- Stimulus process
115  stim_proc: process
116  begin
117
118      -- hold reset state for 100 ns.
119      wait for 10 ns;
120
121      A0 <= NOT A0;    --FLIP VALUES FOR SIMULATION
122      end process;
123
124  stim_proc1: process
125  begin
126      -- hold reset state for 100 ns.
127      wait for 20 ns;    -- DOUBLE WAIT TIME AND REPEAT FOR ALL VALUES
128
129      A1 <= NOT A1;
130      end process;
131
132  stim_proc2: process
133  begin
134      -- hold reset state for 100 ns.
135      wait for 40 ns;

```

The above code is the test bench created showing the input signals A0 to A7 and B, and the output signals Q0 to Q7 which are the result of ANDing signals A0 to A7 with signal B.

```

32 entity LX_8X8 is                                -- SET PORT A INPUTS
33     Port ( A0 : in  STD_LOGIC;
34            A1 : in  STD_LOGIC;
35            A2 : in  STD_LOGIC;
36            A3 : in  STD_LOGIC;
37            A4 : in  STD_LOGIC;
38            A5 : in  STD_LOGIC;
39            A6 : in  STD_LOGIC;
40            A7 : in  STD_LOGIC;
41
42            R1 : in  STD_LOGIC;    --SET REMAINDER INPUTS
43            R2 : in  STD_LOGIC;
44            R3 : in  STD_LOGIC;
45            R4 : in  STD_LOGIC;
46            R5 : in  STD_LOGIC;
47            R6 : in  STD_LOGIC;
48            R7 : in  STD_LOGIC;
49            R8 : in  STD_LOGIC;
50
51            B : in  STD_LOGIC;    -- SET PORT B INPUT
52
53            Q0 : out STD_LOGIC;    -- SET OUTPUTS Q
54            Q1 : out STD_LOGIC;
55            Q2 : out STD_LOGIC;
56            Q3 : out STD_LOGIC;
57            Q4 : out STD_LOGIC;
58            Q5 : out STD_LOGIC;
59            Q6 : out STD_LOGIC;
60            Q7 : out STD_LOGIC;
61            Q8 : out STD_LOGIC
62
63     );
64 end LX_8X8;
65
66 architecture Behavioral of LX_8X8 is
67
68     COMPONENT h_adder                            -- CALL HALF ADDER
69     PORT(
70         A : IN  std_logic;    -- SET INPUTS/OUTPUTS
71         B : IN  std_logic;
72         SUM : OUT std_logic;
73         CARRY : OUT std_logic
74

```

The section of code above sets the inputs and outputs for line x. it also calls in the half adder that was created earlier.

```

75         );
76     END COMPONENT;
77
78     COMPONENT f_adder      -- CALL FULL ADDER
79 PORT(
80     A : IN  std_logic;      -- SET INPUTS/OUTPUTS
81     B : IN  std_logic;
82     CIN : IN  std_logic;
83     SUM : OUT std_logic;
84     CARRY : OUT std_logic
85 );
86 END COMPONENT;
87 SIGNAL A0B : STD_LOGIC; -- ASSIGN SIGNALS AB AS INPUTS
88 SIGNAL A1B : STD_LOGIC;
89 SIGNAL A2B : STD_LOGIC;
90 SIGNAL A3B : STD_LOGIC;
91 SIGNAL A4B : STD_LOGIC;
92 SIGNAL A5B : STD_LOGIC;
93 SIGNAL A6B : STD_LOGIC;
94 SIGNAL A7B : STD_LOGIC;
95
96 SIGNAL C1 : STD_LOGIC; -- ASSIGN SIGNAL C AS CARRY INPUTS
97 SIGNAL C2 : STD_LOGIC;
98 SIGNAL C3 : STD_LOGIC;
99 SIGNAL C4 : STD_LOGIC;
100 SIGNAL C5 : STD_LOGIC;
101 SIGNAL C6 : STD_LOGIC;
102 SIGNAL C7 : STD_LOGIC;
103
104

```

This section calls in the full adder that was previously created and defines the AB signals and C signals as inputs.

```

104
105 begin
106
107
108
109     A0B <= A0 AND B;      -- AND AB SIGNALS WITH A SIGNALS
110     A1B <= A1 AND B;
111     A2B <= A2 AND B;
112     A3B <= A3 AND B;
113     A4B <= A4 AND B;
114     A5B <= A5 AND B;
115     A6B <= A6 AND B;
116     A7B <= A7 AND B;
117
118     HA1 : h_adder          -- IMPLEMENT HALF AND FULL ADDER TO SIGNALS FOR RESULT
119     port map(A0B,R1,Q0,C1);
120
121     FA1 : f_adder
122     PORT MAP (A1B,R2,C1,Q1,C2);
123
124     FA2 : f_adder
125     PORT MAP (A2B,R3,C2,Q2,C3);
126
127     FA3 : f_adder
128     PORT MAP (A3B,R4,C3,Q3,C4);
129
130     FA4 : f_adder
131     PORT MAP (A4B,R5,C4,Q4,C5);
132
133     FA5 : f_adder
134     PORT MAP (A5B,R6,C5,Q5,C6);
135
136     FA6 : f_adder
137     PORT MAP (A6B,R7,C6,Q6,C7);
138
139     FA7 : f_adder
140     PORT MAP (A7B,R8,C7,Q7,Q8);
141
142
143 end Behavioral;

```

This section assigns the values of the signals A0 to A7 ANDed with signal B to create the signal A0B to A7B and then port maps the variables in to the half and full adders.

Like with line 1 a test bench was created to ensure that the component functioned correctly.

The next stage in creating the multiplier was to combine the components line 1 and line x into a MULTI line which became the completed 8 x 8 multiplier.

```

32 entity MULTI_8X8 is
33     Port ( A_0 : in  STD_LOGIC; --DEFINE INPUT BITS PORT A
34           A_1 : in  STD_LOGIC;
35           A_2 : in  STD_LOGIC;
36           A_3 : in  STD_LOGIC;
37           A_4 : in  STD_LOGIC;
38           A_5 : in  STD_LOGIC;
39           A_6 : in  STD_LOGIC;
40           A_7 : in  STD_LOGIC;
41
42
43           B_0 : in  STD_LOGIC; -- DEFINTE  INPUT BITS PORT B
44           B_1 : in  STD_LOGIC;
45           B_2 : in  STD_LOGIC;
46           B_3 : in  STD_LOGIC;
47           B_4 : in  STD_LOGIC;
48           B_5 : in  STD_LOGIC;
49           B_6 : in  STD_LOGIC;
50           B_7 : in  STD_LOGIC;
51
52           Q_0 : out STD_LOGIC; -- DEFINE OUTPUT BITS
53           Q_1 : out STD_LOGIC;
54           Q_2 : out STD_LOGIC;
55           Q_3 : out STD_LOGIC;
56           Q_4 : out STD_LOGIC;
57           Q_5 : out STD_LOGIC;
58           Q_6 : out STD_LOGIC;
59           Q_7 : out STD_LOGIC;
60           Q_8 : out STD_LOGIC;
61           Q_9 : out STD_LOGIC;
62           Q_10: out STD_LOGIC;
63           Q_11: out STD_LOGIC;
64           Q_12: out STD_LOGIC;
65           Q_13: out STD_LOGIC;
66           Q_14: out STD_LOGIC;
67           Q_15: out STD_LOGIC
68     );
69 end MULTI_8X8;

```

This section defined all the inputs and outputs for the MULTI component.

```

70
71 architecture Behavioral of MULTI_8X8 is
72
73     COMPONENT L1_8X8
74     PORT (
75         A0 : IN  std_logic;  --LINE 1 INPUTS PORT A
76         A1 : IN  std_logic;
77         A2 : IN  std_logic;
78         A3 : IN  std_logic;
79         A4 : IN  std_logic;
80         A5 : IN  std_logic;
81         A6 : IN  std_logic;
82         A7 : IN  std_logic;
83         B : IN  std_logic;  -- LINE 1 INPUT PORT B
84         Q0 : OUT std_logic;  -- LINE 1 OUTPUTS
85         Q1 : OUT std_logic;
86         Q2 : OUT std_logic;
87         Q3 : OUT std_logic;
88         Q4 : OUT std_logic;
89         Q5 : OUT std_logic;
90         Q6 : OUT std_logic;
91         Q7 : OUT std_logic;
92     );
93 END COMPONENT;
94
95

```

This section defines the inputs and outputs for line one

```

96     COMPONENT LX_8X8
97     PORT (
98         A0 : IN  std_logic;  --LINE X INPUTS PORT A
99         A1 : IN  std_logic;
100        A2 : IN  std_logic;
101        A3 : IN  std_logic;
102        A4 : IN  std_logic;
103        A5 : IN  std_logic;
104        A6 : IN  std_logic;
105        A7 : IN  std_logic;
106        R1 : IN  std_logic;
107        R2 : IN  std_logic;
108        R3 : IN  std_logic;
109        R4 : IN  std_logic;
110        R5 : IN  std_logic;
111        R6 : IN  std_logic;
112        R7 : IN  std_logic;
113        R8 : IN  std_logic;
114        B : IN  std_logic;  -- LINE X INPUT PORT B
115        Q0 : OUT std_logic;  -- LINE X OUTPUTS
116        Q1 : OUT std_logic;
117        Q2 : OUT std_logic;
118        Q3 : OUT std_logic;
119        Q4 : OUT std_logic;
120        Q5 : OUT std_logic;
121        Q6 : OUT std_logic;
122        Q7 : OUT std_logic;
123        Q8 : OUT std_logic;
124    );
125 END COMPONENT;
126

```

This section defines the inputs and outputs of line x.

```

126
127     SIGNAL R1_1 : STD_LOGIC;    -- ASSIGN VALUES TO REMAINDER SIGNALS|
128     SIGNAL R1_2 : STD_LOGIC;
129     SIGNAL R1_3 : STD_LOGIC;
130     SIGNAL R1_4 : STD_LOGIC;
131     SIGNAL R1_5 : STD_LOGIC;
132     SIGNAL R1_6 : STD_LOGIC;
133     SIGNAL R1_7 : STD_LOGIC;
134
135
136     SIGNAL R2_1 : STD_LOGIC;
137     SIGNAL R2_2 : STD_LOGIC;
138     SIGNAL R2_3 : STD_LOGIC;
139     SIGNAL R2_4 : STD_LOGIC;
140     SIGNAL R2_5 : STD_LOGIC;
141     SIGNAL R2_6 : STD_LOGIC;
142     SIGNAL R2_7 : STD_LOGIC;
143     SIGNAL R2_8 : STD_LOGIC;
144
145     SIGNAL R3_1 : STD_LOGIC;
146     SIGNAL R3_2 : STD_LOGIC;
147     SIGNAL R3_3 : STD_LOGIC;
148     SIGNAL R3_4 : STD_LOGIC;
149     SIGNAL R3_5 : STD_LOGIC;
150     SIGNAL R3_6 : STD_LOGIC;
151     SIGNAL R3_7 : STD_LOGIC;
152     SIGNAL R3_8 : STD_LOGIC;
153
154     SIGNAL R4_1 : STD_LOGIC;
155     SIGNAL R4_2 : STD_LOGIC;
156     SIGNAL R4_3 : STD_LOGIC;
157     SIGNAL R4_4 : STD_LOGIC;
158     SIGNAL R4_5 : STD_LOGIC;
159     SIGNAL R4_6 : STD_LOGIC;
160     SIGNAL R4_7 : STD_LOGIC;
161     SIGNAL R4_8 : STD_LOGIC;
162
163     SIGNAL R5_1 : STD_LOGIC;
164     SIGNAL R5_2 : STD_LOGIC;
165     SIGNAL R5_3 : STD_LOGIC;
166     SIGNAL R5_4 : STD_LOGIC;
167     SIGNAL R5_5 : STD_LOGIC;
168     SIGNAL R5_6 : STD_LOGIC;
169     SIGNAL R5_7 : STD_LOGIC;
170     SIGNAL R5_8 : STD_LOGIC;
171
172     SIGNAL R6_1 : STD_LOGIC;
173     SIGNAL R6_2 : STD_LOGIC;
174     SIGNAL R6_3 : STD_LOGIC;
175     SIGNAL R6_4 : STD_LOGIC;
176     SIGNAL R6_5 : STD_LOGIC;
177     SIGNAL R6_6 : STD_LOGIC;
178     SIGNAL R6_7 : STD_LOGIC;
179     SIGNAL R6_8 : STD_LOGIC;
180
181     SIGNAL R7_1 : STD_LOGIC;
182     SIGNAL R7_2 : STD_LOGIC;
183     SIGNAL R7_3 : STD_LOGIC;
184     SIGNAL R7_4 : STD_LOGIC;
185     SIGNAL R7_5 : STD_LOGIC;
186     SIGNAL R7_6 : STD_LOGIC;
187     SIGNAL R7_7 : STD_LOGIC;
188     SIGNAL R7_8 : STD_LOGIC;
189

```

This section defines the remainder signals for the MULTI component.

```

420
421 begin --PORT MAPS FOR LINE 1-8
422
423 L1 : L1_8X8
424 port map( A_0,A_1,A_2,A_3,A_4,A_5,A_6,A_7,B_0,Q_0,R1_1,R1_2,R1_3,R1_4,R1_5,R1_6,R1_7);
425
426 L2 : L2_8X8
427 port map( A_0,A_1,A_2,A_3,A_4,A_5,A_6,A_7,R1_1,R1_2,R1_3,R1_4,R1_5,R1_6,R1_7,'0' ,B_1,Q_1,R2_1,R2_2,R2_3,R2_4,R2_5,R2_6,R2_7,R2_8);
428
429 L3 : L3_8X8
430 port map( A_0,A_1,A_2,A_3,A_4,A_5,A_6,A_7,R2_1,R2_2,R2_3,R2_4,R2_5,R2_6,R2_7,R2_8,Q_2,R3_1,R3_2,R3_3,R3_4,R3_5,R3_6,R3_7,R3_8);
431
432 L4 : L4_8X8
433 port map( A_0,A_1,A_2,A_3,A_4,A_5,A_6,A_7,R3_1,R3_2,R3_3,R3_4,R3_5,R3_6,R3_7,R3_8,Q_3,R4_1,R4_2,R4_3,R4_4,R4_5,R4_6,R4_7,R4_8);
434
435 L5 : L5_8X8
436 port map( A_0,A_1,A_2,A_3,A_4,A_5,A_6,A_7,R4_1,R4_2,R4_3,R4_4,R4_5,R4_6,R4_7,R4_8,Q_4,R5_1,R5_2,R5_3,R5_4,R5_5,R5_6,R5_7,R5_8);
437
438 L6 : L6_8X8
439 port map( A_0,A_1,A_2,A_3,A_4,A_5,A_6,A_7,R5_1,R5_2,R5_3,R5_4,R5_5,R5_6,R5_7,R5_8,Q_5,R6_1,R6_2,R6_3,R6_4,R6_5,R6_6,R6_7,R6_8);
440
441 L7 : L7_8X8
442 port map( A_0,A_1,A_2,A_3,A_4,A_5,A_6,A_7,R6_1,R6_2,R6_3,R6_4,R6_5,R6_6,R6_7,R6_8,Q_6,R7_1,R7_2,R7_3,R7_4,R7_5,R7_6,R7_7,R7_8);
443
444 L8 : L8_8X8
445 port map( A_0,A_1,A_2,A_3,A_4,A_5,A_6,A_7,R7_1,R7_2,R7_3,R7_4,R7_5,R7_6,R7_7,R7_8,Q_7,Q_8,Q_9,Q_10,Q_11,Q_12,Q_13,Q_14,Q_15);
446
447 end Behavioral;
448
449

```

This section shows the port mapping of line 1 and line x, line x is then used for L3 to L8 to create the final 8 x 8 multiplier.

```

37
38 ARCHITECTURE behavior OF MULTI_8_8_TB IS
39
40     -- Component Declaration for the Unit Under Test (UUT)
41
42     COMPONENT MULTI_8X8
43     PORT (
44         A_0 : IN    std_logic;    -- SET A SIGNALS AND B SIGNALS AS INPUTS
45         A_1 : IN    std_logic;
46         A_2 : IN    std_logic;
47         A_3 : IN    std_logic;
48         A_4 : IN    std_logic;
49         A_5 : IN    std_logic;
50         A_6 : IN    std_logic;
51         A_7 : IN    std_logic;
52         B_0 : IN    std_logic;
53         B_1 : IN    std_logic;
54         B_2 : IN    std_logic;
55         B_3 : IN    std_logic;
56         B_4 : IN    std_logic;
57         B_5 : IN    std_logic;
58         B_6 : IN    std_logic;
59         B_7 : IN    std_logic;
60         Q_0 : OUT   std_logic;    -- SET Q SIGNALS AS OUTPUTS
61         Q_1 : OUT   std_logic;
62         Q_2 : OUT   std_logic;
63         Q_3 : OUT   std_logic;
64         Q_4 : OUT   std_logic;
65         Q_5 : OUT   std_logic;
66         Q_6 : OUT   std_logic;
67         Q_7 : OUT   std_logic;
68         Q_8 : OUT   std_logic;
69         Q_9 : OUT   std_logic;
70         Q_10 : OUT  std_logic;
71         Q_11 : OUT  std_logic;
72         Q_12 : OUT  std_logic;
73         Q_13 : OUT  std_logic;
74         Q_14 : OUT  std_logic;
75         Q_15 : OUT  std_logic;
76     );
77     END COMPONENT;
78

```

This section shows the input and output assignment of the signals for the test bench of the MULTI component.

```

80  --Inputs
81  signal A_0 : std_logic := '0';    -- SET INPUT SIGNALS TO 0
82  signal A_1 : std_logic := '0';
83  signal A_2 : std_logic := '0';
84  signal A_3 : std_logic := '0';
85  signal A_4 : std_logic := '0';
86  signal A_5 : std_logic := '0';
87  signal A_6 : std_logic := '0';
88  signal A_7 : std_logic := '0';
89  signal B_0 : std_logic := '0';
90  signal B_1 : std_logic := '0';
91  signal B_2 : std_logic := '0';
92  signal B_3 : std_logic := '0';
93  signal B_4 : std_logic := '0';
94  signal B_5 : std_logic := '0';
95  signal B_6 : std_logic := '0';
96  signal B_7 : std_logic := '0';
97
98  --Outputs
99  signal Q_0 : std_logic;    -- SET OUTPUT SIGNALS
100 signal Q_1 : std_logic;
101 signal Q_2 : std_logic;
102 signal Q_3 : std_logic;
103 signal Q_4 : std_logic;
104 signal Q_5 : std_logic;
105 signal Q_6 : std_logic;
106 signal Q_7 : std_logic;
107 signal Q_8 : std_logic;
108 signal Q_9 : std_logic;
109 signal Q_10 : std_logic;
110 signal Q_11 : std_logic;
111 signal Q_12 : std_logic;
112 signal Q_13 : std_logic;
113 signal Q_14 : std_logic;
114 signal Q_15 : std_logic;
115 -- No clocks detected in port list. Replace <clock> below with
116 -- appropriate port name
117
118
119

```

This shows the input signals being set to 0 and sets the output signals.

```

119
120 BEGIN
121
122     -- Instantiate the Unit Under Test (UUT)
123     uut: MULTI_8X8 PORT MAP (          -- SET POT MAP 8X8
124         A_0 => A_0,
125         A_1 => A_1,                      -- ASSIGN VALUES AS THEMSELVES
126         A_2 => A_2,
127         A_3 => A_3,
128         A_4 => A_4,
129         A_5 => A_5,
130         A_6 => A_6,
131         A_7 => A_7,
132         B_0 => B_0,
133         B_1 => B_1,
134         B_2 => B_2,
135         B_3 => B_3,
136         B_4 => B_4,
137         B_5 => B_5,
138         B_6 => B_6,
139         B_7 => B_7,
140         Q_0 => Q_0,
141         Q_1 => Q_1,
142         Q_2 => Q_2,
143         Q_3 => Q_3,
144         Q_4 => Q_4,
145         Q_5 => Q_5,
146         Q_6 => Q_6,
147         Q_7 => Q_7,
148         Q_8 => Q_8,
149         Q_9 => Q_9,
150         Q_10 => Q_10,
151         Q_11 => Q_11,
152         Q_12 => Q_12,
153         Q_13 => Q_13,
154         Q_14 => Q_14,
155         Q_15 => Q_15
156     );
157
158
159
160

```

This section begins the testing of the component by assigning all the signal values as themselves.

```

163 stim_proc0: process    --START SIMULATION PROCESS AND FLIP VALUES FOR TEST A SIGNALS
164 begin
165     wait for 10 ns;
166     A_0 <= NOT A_0;
167 end process;
168
169 stim_proc1: process
170 begin
171     wait for 20 ns;
172     A_1 <= NOT A_1;
173 end process;
174
175 stim_proc2: process
176 begin
177     wait for 40 ns;
178     A_2 <= NOT A_2;
179 end process;
180
181 stim_proc3: process
182 begin
183     wait for 80 ns;
184     A_3 <= NOT A_3;
185 end process;
186
187 stim_proc4: process
188 begin
189     wait for 160 ns;
190     A_4 <= NOT A_4;
191 end process;
192
193 stim_proc5: process
194 begin
195     wait for 320 ns;
196     A_5 <= NOT A_5;
197 end process;
198
199 stim_proc6: process
200 begin
201     wait for 640 ns;
202     A_6 <= NOT A_6;
203 end process;
204
205 stim_proc7: process
206 begin
207     wait for 1280 ns;
208     A_7 <= NOT A_7;
209 end process;
210
211
212 -- PORT B
213 stim_proc10: process    --START SIMULATION PROCESS AND FLIP VALUES FOR TEST B SIGNALS
214 begin
215     wait for 2560 ns;
216     B_0 <= NOT B_0;
217 end process;
218
219 stim_proc11: process
220 begin
221     wait for 5120 ns;
222     B_1 <= NOT B_1;
223 end process;
224
225 stim_proc12: process
226 begin
227     wait for 10240 ns;
228     B_2 <= NOT B_2;
229 end process;
230
231 stim_proc13: process
232 begin
233     wait for 20480 ns;
234     B_3 <= NOT B_3;
235 end process;
236
237 stim_proc14: process
238 begin
239     wait for 40960 ns;
240     B_4 <= NOT B_4;
241 end process;
242
243 stim_proc15: process
244 begin
245     wait for 81920 ns;
246     B_5 <= NOT B_5;
247 end process;
248
249 stim_proc16: process
250 begin
251     wait for 163840 ns;
252     B_6 <= NOT B_6;
253 end process;
254
255 stim_proc17: process
256 begin
257     wait for 327680 ns;
258     B_7 <= NOT B_7;
259 end process;

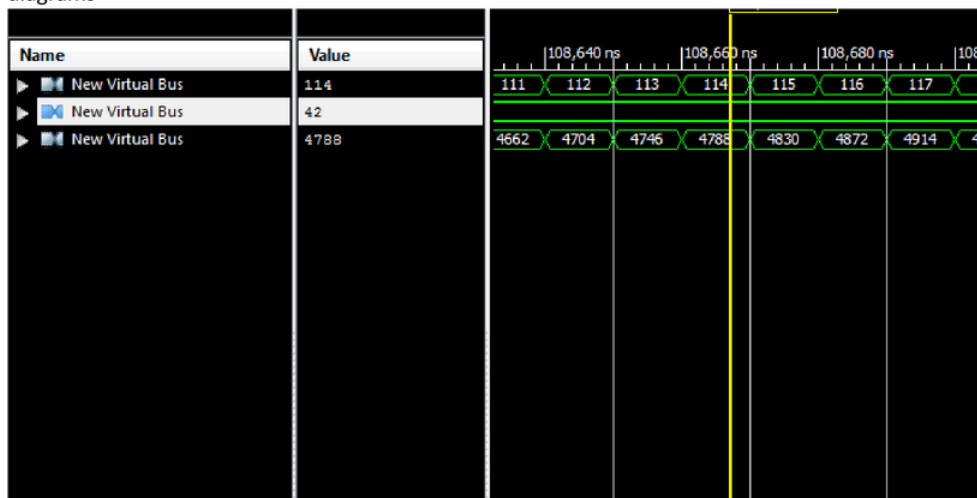
```

The preceding diagrams show the simulation process used in the test bench which flips the values of the signals for testing and includes the time delay function between each signal being tested. The time delays, as with the previous test benches for line 1 and line x double with each consecutive signal test.

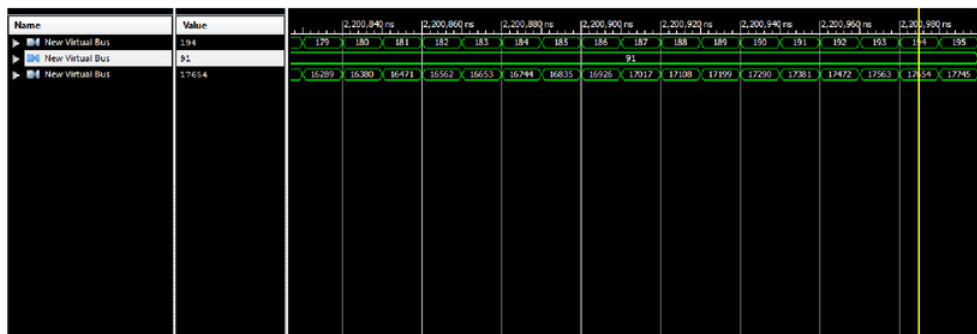
Once the test bench had been checked for syntax errors it was simulated to ensure that it was functioning properly.



As can be seen in the diagram above the multiplier worked as expected with a binary value of 31 being multiplied by a binary value of 34 giving a correct result of 1054. This was checked again with several different values to ensure that it was functioning correctly as can be seen in the next two diagrams



Again, the values 114 and 42 when multiplied are 4788 providing more proof that the multiplier functions correctly.



This final diagram shows that the multiplier functions correctly with the values 194 multiplied by 91 giving the correct result of 17654.

5. conclusion

the design of the 8 x 8 multiplier and each of its components was successful and was demonstrated to work correctly. At first arrays were for the variables but for some, unknown reason it would not work so the lines 1 and x were rebuilt without the arrays and were found to function correctly. For future work using VHDL and the Xilinx design software a better understanding would be needed but for this assignment it is believed that the level of comprehension of the language and software was sufficient enough to perform the required task.

6. references

- [1] M. Thakur and J. Ashraf, "Design of Braun Multiplier with Kogge Stone Adder & It's Implementation on FPGA," *Int. J. Sci. Eng. Res.*, vol. 3, no. 10, 2012.
- [2] "What is an FPGA? What is an ASIC?" [Online]. Available: <https://www.nandland.com/articles/what-is-an-fpga-what-is-an-asic.html>. [Accessed: 11-May-2018].
- [3] "VHDL Primer." [Online]. Available: https://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html. [Accessed: 11-May-2018].

FINAL GRADE

GENERAL COMMENTS

Instructor

46/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

GRADING FORM: SMU GENERIC_GJ

DANIEL LENNON**46.10****RESEARCH-INFORMED LITERATURE**

Virtually no research on other multipliers designs, could have been vastly improved with other types and a comparison table. 25%

2.5**KNOWLEDGE AND UNDERSTANDING**

You have demonstrated knowledge with your VHDL code. Your code carries few remarks which makes it difficult to follow. 52%

15.6**ANALYSIS**

You have demonstrated the testbench of your design at each stage but have failed to comment on each section. There are no explanations of operation. 50%

10**PRACTICAL APPLICATION AND DEP.**

You have included limited simulation results, demonstrating the operation of the circuit but more images and discussion would have been welcome. Sections of the design has been tested before

13.5

reuse in the next section but there is no proof of operation of each sections as there are no waveforms. 45%

SKILLS FOR PROFESSIONAL PRAC.



The report is poorly presented with some good diagrams and illustrations. A small bibliography and reference section is included. 45%

4.5