

8x8 Ryan O'Sullivan

by Ryan O'Sullivan

FILE	3258_RYAN_OSULLIVAN_8X8_RYAN_OSULLIVAN_114255_1250705814.D OCX		
TIME SUBMITTED	11-MAY-2018 12:35PM (UTC+0100)	WORD COUNT	2590
SUBMISSION ID	88975010	CHARACTER COUNT	11972



PRIFYSGOL CYMRU

Y Drindod Dewi Sant

UNIVERSITY OF WALES

Trinity Saint David

<8-bit x 8-bit unsigned number multiplier>

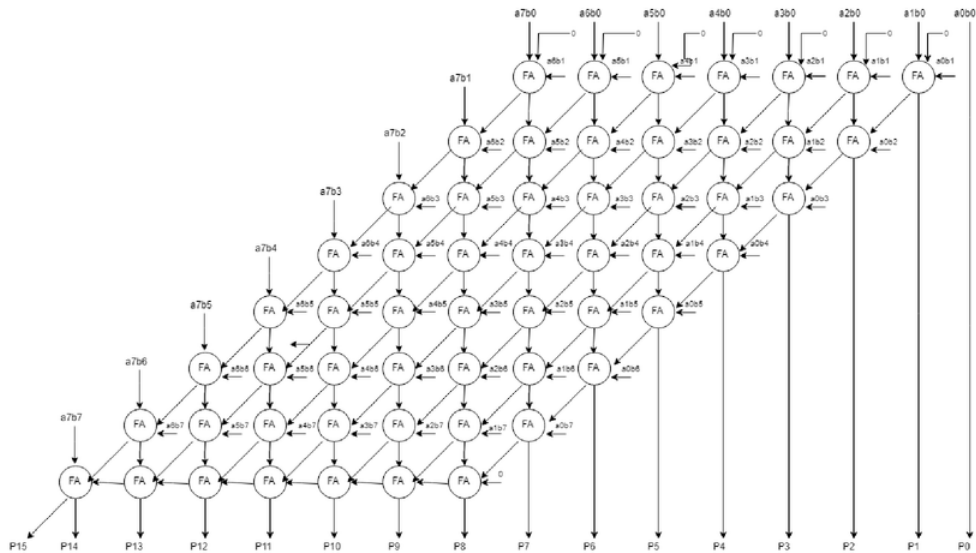
<Ryan O'Sullivan>

<BEng Electronics and computing>

<Gary Jones>

Introduce

This report will synthesise an 8-bit x 8-bit unsigned number multiplier based upon a simple parallel Braun multiplier unsigned number this will give the ability to multiply 8 numbers by 8 numbers.

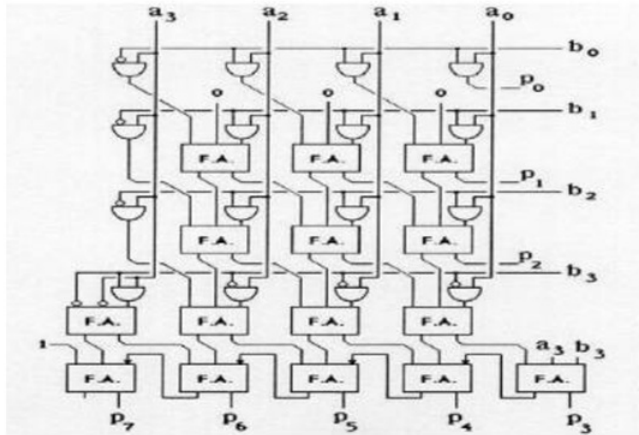


The above image is an 8x8 multiplier based upon the Braun architectures this will be used to help create the advance VHDL design code that will be implemented in Xilinx this design will use Half adders and a full adder to create this design only using AND gates.

The final product of this report will be able to Multiply 8 binary numbers by 8 binary numbers this in decimal will only allowed multiplication of $255 \times 255 = 65025$ this will be the maximum calculation this device will be able to produce.

Braun multiplier

The Braun multiplier Architecture (carry multiplier) is a parallel multiplier the main structure of this Architecture is built of AND gates and adders these are used in an iterative manner. This system is widely used being as it is such an effective Multiplier. the Braun multiplier can have issues with glitches, but this is rare. This performs well for the unsigned operands that are under 16 bits over this and errors can occur.



This show a 4x4 Braun multiplier as you can see A is the

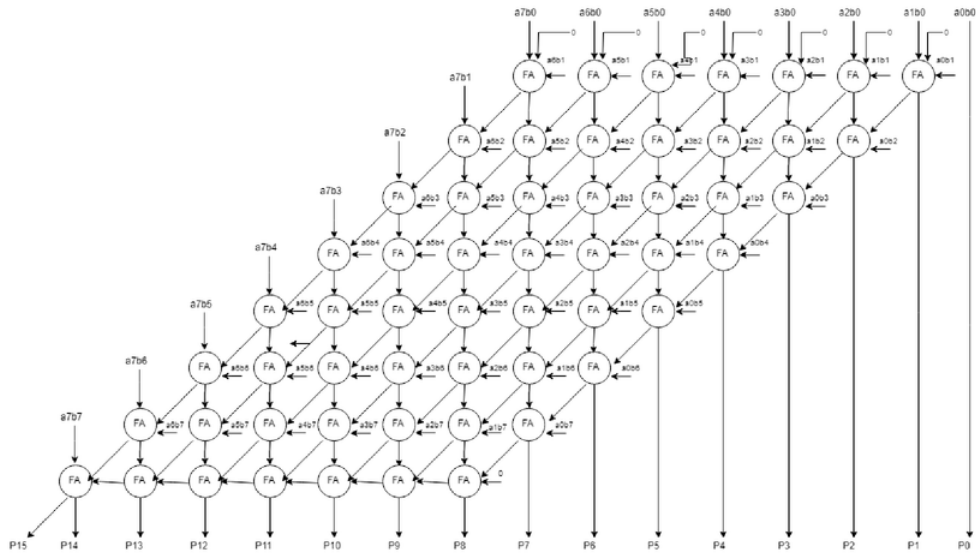
where b is the 4-bit multiplicand and a is the 4-bit multiplier. P is the final result of the values.

A0	0	B0	1
A1	1	B1	0
A2	1	B2	0
A3	0	B3	1

So here we are multiplying $0110 \times 1001 = 110110$

What is an 8x8 Architecture

The 8x8 Architecture is completely designed using gates and adders the 8x8 design will be based on the Braun architecture the size will be 8 bits wide and 8 bits deep the flow of this 8x8 design is shown in the introduction. This Diagram was specifically made for this report and outline in greater detail the flow of the 8x8 architecture it was created using Draw.io With background knowledge of the 8x8 Architecture this diagram helped greatly when Coding in VHDL



In this Diagram, P is the final output A and AB are the inputs.

What is VHDL

VHDL (VHSIC Hardware Description Language) is a standard language that is used for describing a structure and function for an integrated circuit this has been an IEEE standard since 1987 used in electronic design automation allowing for the creation of field-programmable gate arrays.

Line one

Line one is the first input of the 8x8 array where input 0-7B these values will be injected through an and gates with the X0 value. This will create output 0-7Q these values will then be flown into the line X.

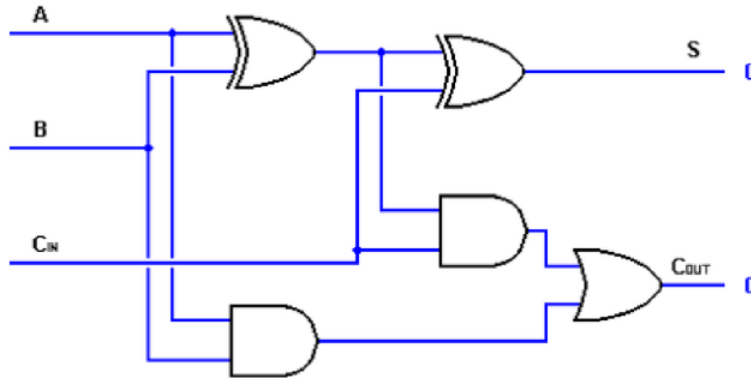
The X lines

Line X will be the structure used for rest for the rest of the Architecture it will consist of one half adder and 7 Full adders. The full adder will be able to add the carry on their inputs and carry on the output. There will be 8 outputs from line 1 that will be inputted to line x. there will be another 8 input from line1 for the previous signal all adders will give an output of Q1-Q7 this process will repeat.

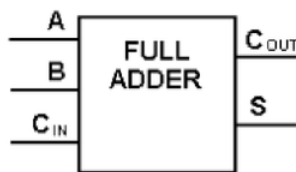
Full Adder

Full adder will be one of the main component used in this assignment the truth table for the full adder as shown in figure * shows the operation of the full adder from this

The full adder diagram shown below is comprised of OR gates, 2 XOR gates, 2 AND gates Cout will be the carry A and B will be the inputs S will be the sum That can be broken down to 2 Half adders this combined with an OR gate can give the Cout results.



This show a simple diagram showing the input into the full adder and outputs.



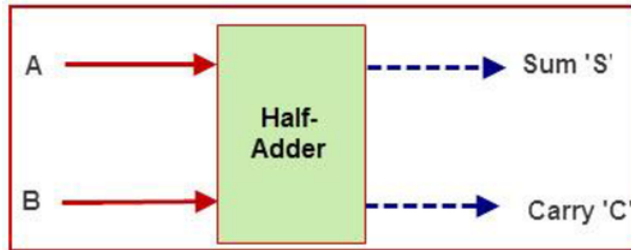
INPUTS		OUTPUTS		
A	B	C _{IN}	C _{OUT}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

From the above truth table it is able to observe the pattern of the full adder this is similar to the half adder as the same results are shadowed on the Truth table The full adder gives the ability to Add the C_{in} to the A, B inputs. Therefore, the carry input is the additional input for the full adder.

Half Adder

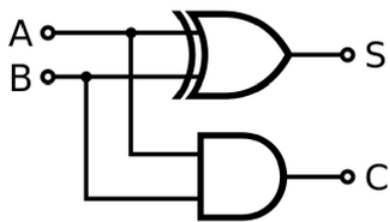
A half adder is able to add two one-bit binary number together and send out a carry if one needs a half add is not able to receive a carry. if you take 2 half adders with an Or gate you would have a full adder. A half adder is used in

Adder come in a load of variation that is commonly used the report will be focusing on only two adder types the half adder and the full adder a half adder is probably the most simple of all the adder with the smallest amount of gates. It has 2 outputs(sum and carry) and 2 input.



The truth table for this Half adder is shown below this shows that only if input A and B are 1 then the Cout will be 1. The half adder gate layout is also shown below.

A (input)	B (input)	C (output)	S (output)
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0



Implementation

After researching the need areas it was then possible to advance the understanding of VHDL operations and applying these clear approach to programming an 8x8 multipliers. the system that is going to be used to build the VHDL code will be Xilinx ISE Design Suite. The coding for 8x8 more multiple will be done in sections half adder, full adder, line one and line X these will be together in the last section the multi8x8. The multi-file will produce the final results in test bench to confirm if final programming is correctly operating all code been annotated to explain the flow of programming.

Full adder

VDHL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity fulladder is
    Port ( A : in  STD_LOGIC;
          B : in  STD_LOGIC;
          Cin : in  STD_LOGIC;
          Sum : out STD_LOGIC;
          Carry : out STD_LOGIC);
end fulladder;

architecture Behavioral of fulladder is

begin

    Sum <= A xor B xor Cin;
    Carry <= (A and b) or ((A xor B) and Cin);

end Behavioral;
```

Test bench VHDL

```
BEGIN

-- Instantiate the Unit Under Test (UUT)
uut: fulladder PORT MAP (
    A => A,
    B => B,
    Cin => Cin,
    Sum => Sum,
    Carry => Carry
);

-- Stimulus process
stim_proc: process
begin
    -- hold reset state for 100 ns.
    wait for 10 ns;
    A <= not A;

end process;

stim_proc1: process
begin
    -- hold reset state for 100 ns.
    wait for 20 ns;
    B <= not B;

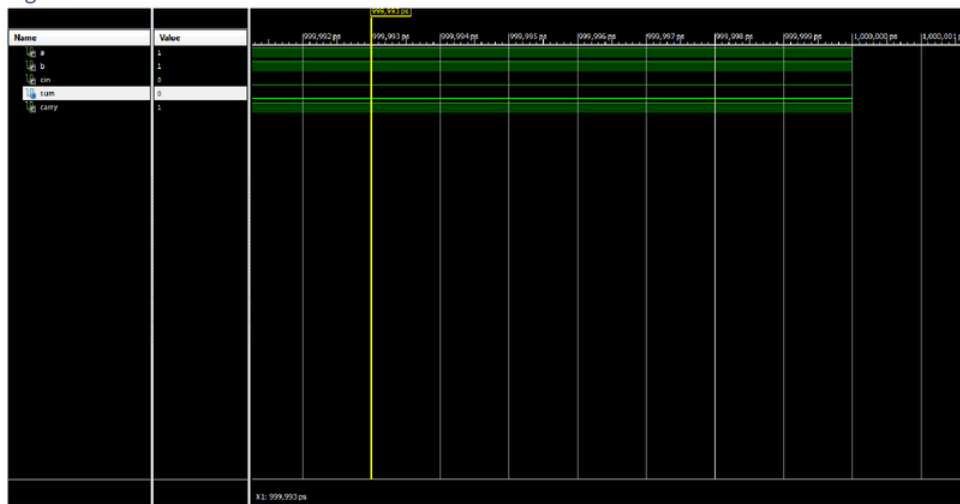
end process;

stim_proc2: process
begin
    -- hold reset state for 100 ns.
    wait for 40 ns;
    Cin <= not Cin;

end process;

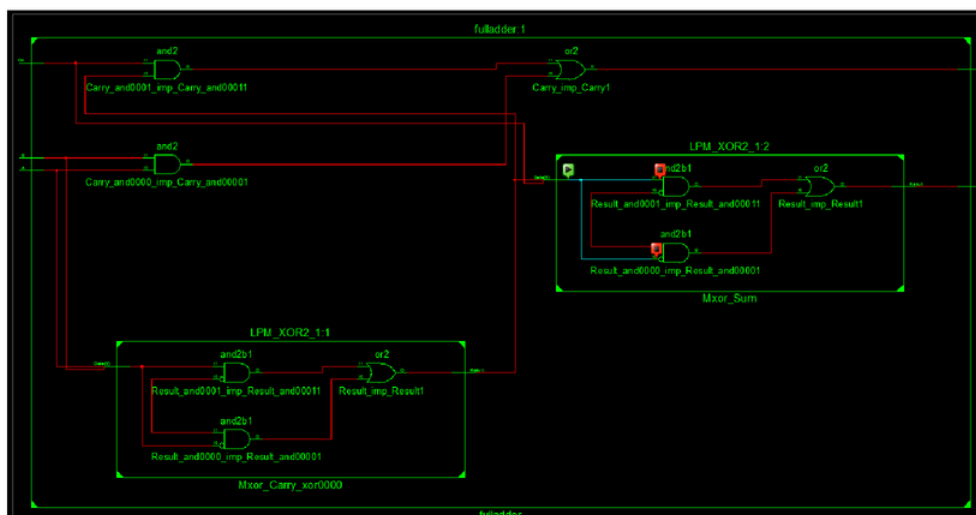
END;
```

Signal



Logic Gates

This shows the logical structure of a Full adder. It is possible to see that the adder is created with 6 AND gates and 3 OR gates.



Half adder

VDHL

```
35 ENTITY Halfadde_tb IS
36 END Halfadde_tb;
37
38 ARCHITECTURE behavior OF Halfadde_tb IS
39
40     -- Component Declaration for the Unit Under Test (UUT)
41
42     COMPONENT Halfadder
43     PORT(
44         A : IN  std_logic;
45         B : IN  std_logic;
46         SUM : OUT std_logic;
47         Carry : OUT std_logic
48     );
49     END COMPONENT;
50
51
52     --Inputs
53     signal A : std_logic := '0';
54     signal B : std_logic := '0';
55
56     --Outputs
57     signal SUM : std_logic;
58     signal Carry : std_logic;
59     -- No clocks detected in port list. Replace <clock> below with
60     -- appropriate port name
61
62
63 BEGIN
64
65     -- Instantiate the Unit Under Test (UUT)
66     uut: Halfadder PORT MAP (
67         A => A,
68         B => B,
69         SUM => SUM,
70         Carry => Carry
71     );
72
73
74
75     -- Stimulus process
76     stim_proc: process
77     begin
78         -- hold reset state for 100 ns.
79         wait for 10 ns;
80         A <= not A;
81     end process;
```

Test bench VHDL

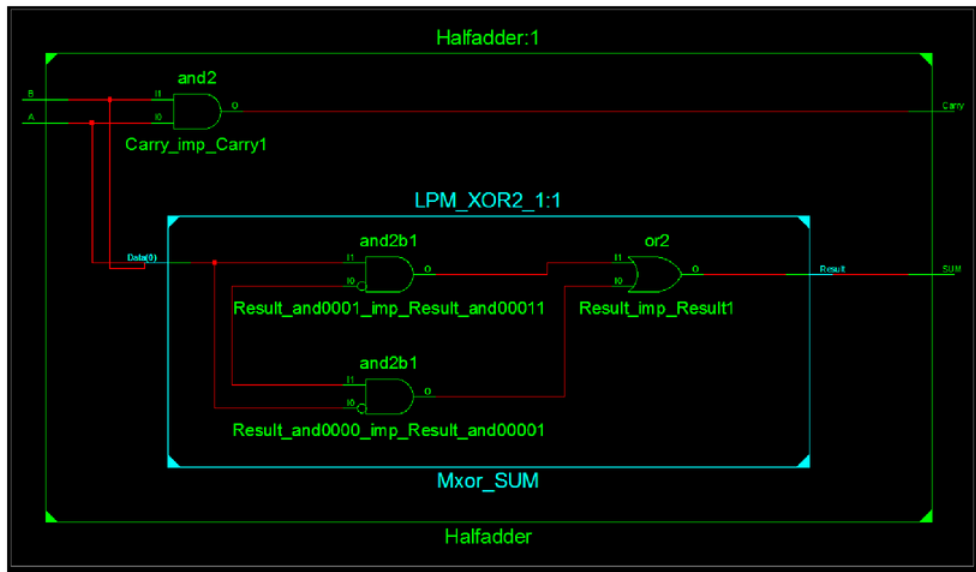
```

35 ENTITY Halfadde_tb IS
36 END Halfadde_tb;
37
38 ARCHITECTURE behavior OF Halfadde_tb IS
39
40     -- Component Declaration for the Unit Under Test (UUT)
41
42     COMPONENT Halfadder
43     PORT(
44         A : IN  std_logic;
45         B : IN  std_logic;
46         SUM : OUT std_logic;
47         Carry : OUT std_logic
48     );
49     END COMPONENT;
50
51
52     --Inputs
53     signal A : std_logic := '0';
54     signal B : std_logic := '0';
55
56     --Outputs
57     signal SUM : std_logic;
58     signal Carry : std_logic;
59     -- No clocks detected in port list. Replace <clock> below with
60     -- appropriate port name
61
62 BEGIN
63
64     -- Instantiate the Unit Under Test (UUT)
65     uut: Halfadder PORT MAP (
66         A => A,
67         B => B,
68         SUM => SUM,
69         Carry => Carry
70     );
71
72
73
74
75     -- Stimulus process
76     stim_proc: process
77     begin
78         -- hold reset state for 100 ns.
79         wait for 10 ns;
80         A <= not A;
81     end process;
82
83     stim_procl: process
84     begin
85         wait for 20 ns;
86         B <= not B;
87     end process;

```

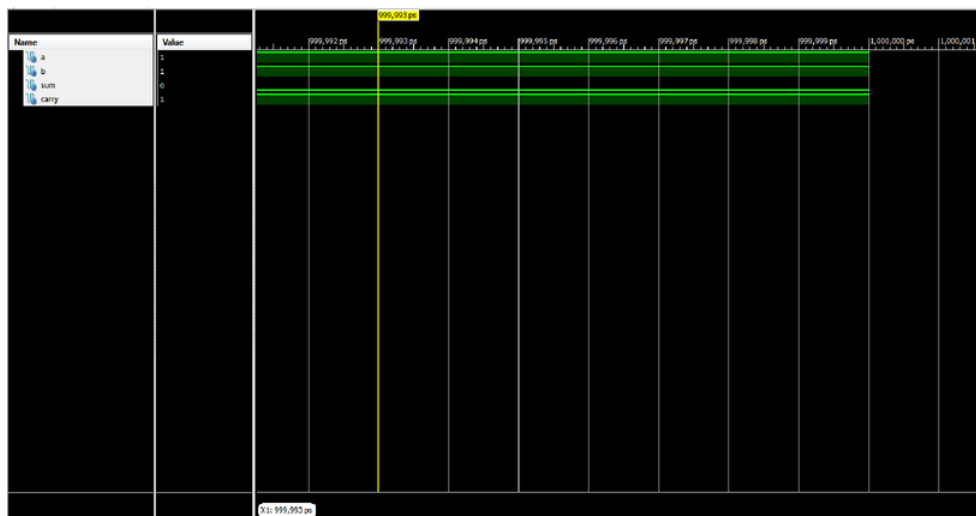
Logic Gates

This shows the logic gate configuration of the Half adder here you can see only created from AND Logic Gates.



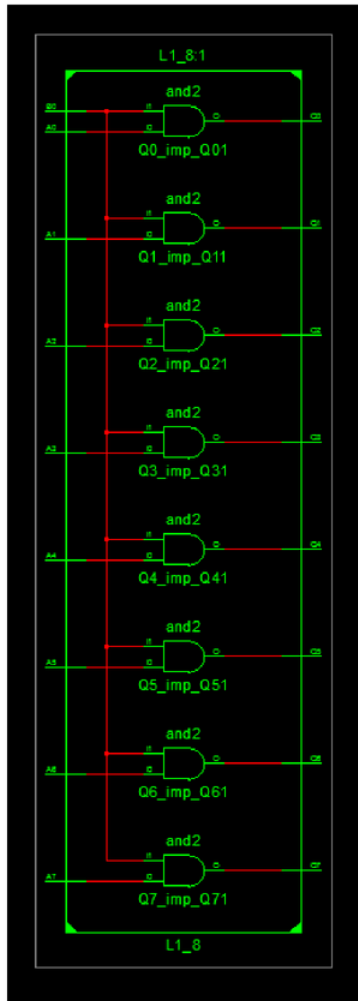
Signal

This shows the signal from the half adder as you can see this does not contain a carry input.



Line one

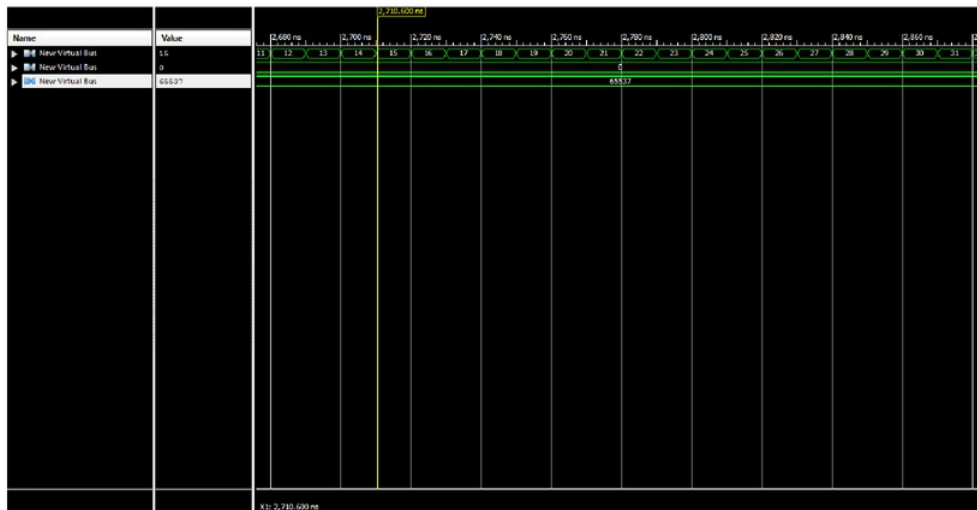
Line one contains the first lot of gates for the 8x8 multiplier this is shown in the below diagram this contains 8 AND gates the input for these gates are A0-A7 and B0 these will give output from Q0- Q7 these output results will flow in to the Line X. Line one will be the first instruction to be used in the VDHL.



Seeing all the gates lined up to give a better understanding of the working of the 8x8 multiple first lines would look if made in real life.

Signal

This show the signal from line 1 from this you can see line one function dealing with the first lot of signal produced. As line one is only made up of And gates the output will always be 0 until both inputs of A and B are = 1



VDHL code

The VDHL code shows all the assigned output/inputs for the behavioural logic A0-A7 are inputs AND B with an output of Q0-Q7 all code been animated to explain each element.

```
31
32 entity L1_8 is
33     Port ( A0 : in  STD_LOGIC;  --INPUT STANDARD LOGIC
34           A1 : in  STD_LOGIC;  --INPUT STANDARD LOGIC
35           A2 : in  STD_LOGIC;  --INPUT STANDARD LOGIC
36           A3 : in  STD_LOGIC;  --INPUT STANDARD LOGIC
37           A4 : in  STD_LOGIC;  --INPUT STANDARD LOGIC
38           A5 : in  STD_LOGIC;  --INPUT STANDARD LOGIC
39           A6 : in  STD_LOGIC;  --INPUT STANDARD LOGIC
40           A7 : in  STD_LOGIC;  --INPUT STANDARD LOGIC
41           B0 : in  STD_LOGIC;  --INPUT STANDARD LOGIC
42           Q0 : out STD_LOGIC;  --OUT STANDARD LOGIC
43           Q1 : out STD_LOGIC;  --OUT STANDARD LOGIC
44           Q2 : out STD_LOGIC;  --OUT STANDARD LOGIC
45           Q3 : out STD_LOGIC;  --OUT STANDARD LOGIC
46           Q4 : out STD_LOGIC;  --OUT STANDARD LOGIC
47           Q5 : out STD_LOGIC;  --OUT STANDARD LOGIC
48           Q6 : out STD_LOGIC;  --OUT STANDARD LOGIC
49           Q7 : out STD_LOGIC);  --OUT STANDARD LOGIC
50 end L1_8;
51
52 architecture Behavioral of L1_8 is
53
54 begin
55     Q0 <= A0 AND B0;  --Q0 IS A0 AND B0
56     Q1 <= A1 AND B0;  --Q1 IS A1 AND B0
57     Q2 <= A2 AND B0;  --Q2 IS A2 AND B0
58     Q3 <= A3 AND B0;  --Q3 IS A3 AND B0
59     Q4 <= A4 AND B0;  --Q4 IS A4 AND B0
60     Q5 <= A5 AND B0;  --Q5 IS A5 AND B0
61     Q6 <= A6 AND B0;  --Q6 IS A6 AND B0
62     Q7 <= A7 AND B0;  --Q7 IS A7 AND B0
63
64 end Behavioral;
65
66
```

Line X

Line X will be used for all line apart from line one this will create the outline for the rest of 8x8 multiple it will consist of 7 full adders and one-half adder this layout will be used for line 2-8 the is done with a series of adders. Line X is very heavy one coding this is cos of there are more interactions with adders.

VDHL code

```
entity L1_8X is
  Port ( A0 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        A1 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        A2 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        A3 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        A4 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        A5 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        A6 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        A7 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        B  : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        Q0 : out STD_LOGIC;      --SET OUTPUT STANDARD LOGIC
        Q1 : out STD_LOGIC;      --SET OUTPUT STANDARD LOGIC
        Q2 : out STD_LOGIC;      --SET OUTPUT STANDARD LOGIC
        Q3 : out STD_LOGIC;      --SET OUTPUT STANDARD LOGIC
        Q4 : out STD_LOGIC;      --SET OUTPUT STANDARD LOGIC
        Q5 : out STD_LOGIC;      --SET OUTPUT STANDARD LOGIC
        Q6 : out STD_LOGIC;      --SET OUTPUT STANDARD LOGIC
        Q7 : out STD_LOGIC;      --SET OUTPUT STANDARD LOGIC
        Q8 : out STD_LOGIC;      --SET OUTPUT STANDARD LOGIC
        X1 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        X2 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        X3 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        X4 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        X5 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        X6 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        X7 : in  STD_LOGIC;      --SET INPUT STANDARD LOGIC
        X8 : in  STD_LOGIC);     --SET INPUT STANDARD LOGIC
end L1_8X;

architecture Behavioral of L1_8X is

  COMPONENT Halfadder
    PORT(
      A : IN  std_logic;
      B : IN  std_logic;
      SUM : OUT std_logic;
      Carry : OUT std_logic
    );
  END COMPONENT;


```

This show all the input/out stated in line X and setting them to standard logic. With all the input setting Port mapping these input to the need gates was needed.

```

81
82 SIGNAL A0B : std_logic; --SETTING SIGNAL AS INPUT
83 SIGNAL A1B : std_logic; --SETTING SIGNAL AS INPUT
84 SIGNAL A2B : std_logic; --SETTING SIGNAL AS INPUT
85 SIGNAL A3B : std_logic; --SETTING SIGNAL AS INPUT
86 SIGNAL A4B : std_logic; --SETTING SIGNAL AS INPUT
87 SIGNAL A5B : std_logic; --SETTING SIGNAL AS INPUT
88 SIGNAL A6B : std_logic; --SETTING SIGNAL AS INPUT
89 SIGNAL A7B : std_logic; --SETTING SIGNAL AS INPUT
90
91 SIGNAL C1 : std_logic; --SETTING SIGNAL AS INPUT/OUTPUT
92 SIGNAL C2 : std_logic; --SETTING SIGNAL AS INPUT/OUTPUT
93 SIGNAL C3 : std_logic; --SETTING SIGNAL AS INPUT/OUTPUT
94 SIGNAL C4 : std_logic; --SETTING SIGNAL AS INPUT/OUTPUT
95 SIGNAL C5 : std_logic; --SETTING SIGNAL AS INPUT/OUTPUT
96 SIGNAL C6 : std_logic; --SETTING SIGNAL AS INPUT/OUTPUT
97 SIGNAL C7 : std_logic; --SETTING SIGNAL AS INPUT/OUTPUT
98
99 begin
100
101     A0B <= A0 AND B; --A0B IS A0 AND B
102     A1B <= A1 AND B; --A1B IS A1 AND B
103     A2B <= A2 AND B; --A2B IS A2 AND B
104     A3B <= A3 AND B; --A3B IS A3 AND B
105     A4B <= A4 AND B; --A4B IS A4 AND B
106     A5B <= A5 AND B; --A5B IS A5 AND B
107     A6B <= A6 AND B; --A6B IS A6 AND B
108     A7B <= A7 AND B; --A7B IS A7 AND B
109
110     HA1 : Halfadder
111         port map (A0B,X1,Q0,C1); -- PORT MAP HALF ADDER
112
113     FA1 : fulladder --PORT MAP FULL ADDDER
114         PORT MAP (A1B , X2 , C1 , Q1 , C2 );
115
116     FA2 : fulladder --PORT MAP FULL ADDDER
117         PORT MAP (A2B , X3 , C2 , Q2 , C3);
118
119     FA3 : fulladder --PORT MAP FULL ADDDER
120         PORT MAP (A3B , X4 , C3 , Q3 , C4);
121
122     FA4 : fulladder --PORT MAP FULL ADDDER
123         PORT MAP (A4B , X5 , C4 , Q4 , C5);
124
125     FA5 : fulladder --PORT MAP FULL ADDDER
126         PORT MAP (A5B , X6 , C5 , Q5 , C6);
127
128     FA6 : fulladder --PORT MAP FULL ADDDER
129         PORT MAP (A6B , X7 , C6 , Q6 , C7);
130
131     FA7 : fulladder --PORT MAP FULL ADDDER
132         PORT MAP (A7B , X8 , C7 , Q7 , Q8);
133
134 end Behavioral;

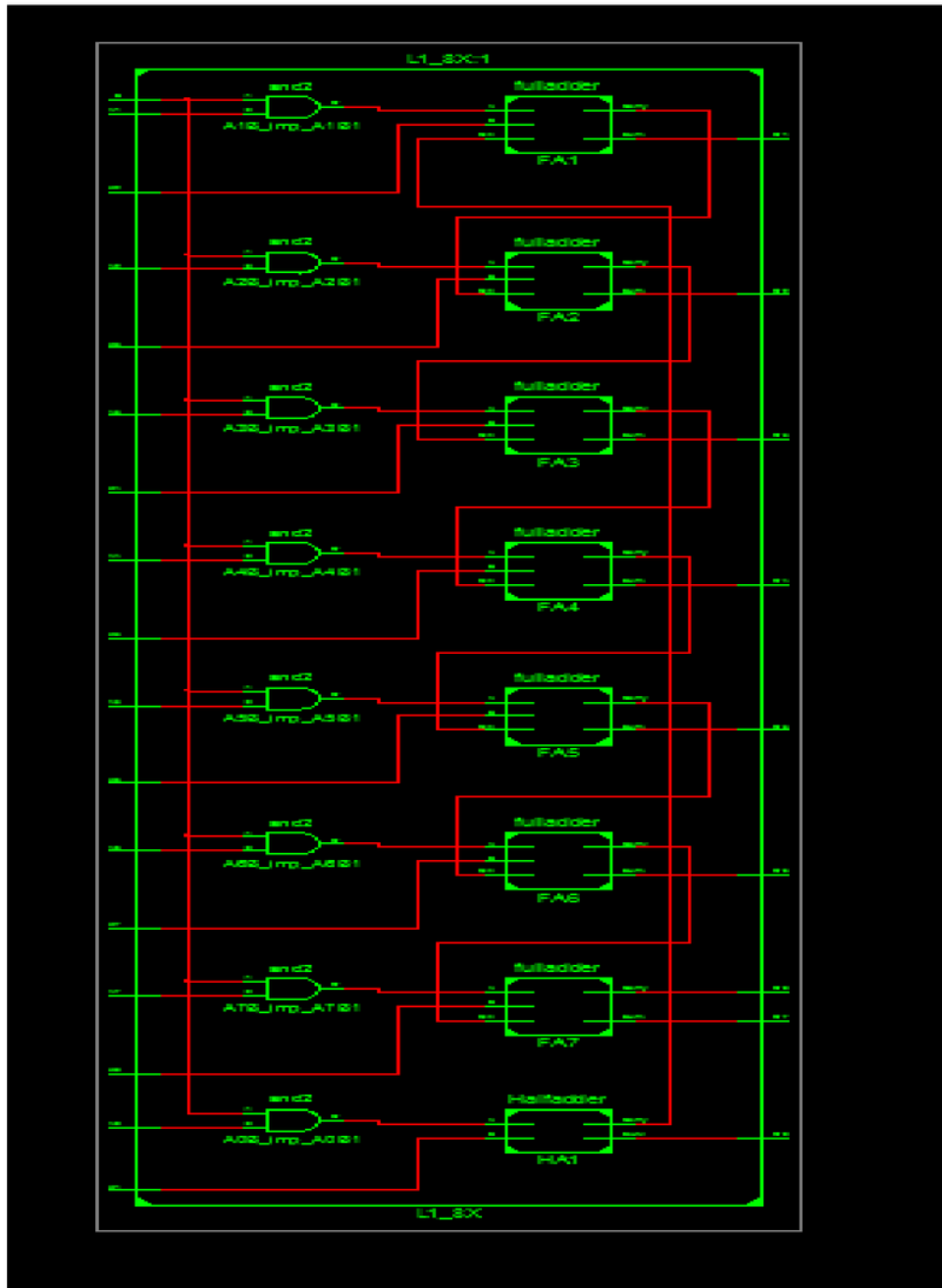
```

Port mapping for this device is done by stating inputs on the left with outputs of the right,

As shown here 1HA(half adder and 7full adders are used in this VHDL coding. An example of the port mapping is H1 inputs are A0B, X1 and the outputs are Q0, C1

Adder and Gate diagram

Here you can see Line X build up of Adders and Gates this diagram help understand the flow of this coding in components. On the left are all inputs(A-B) and an output of Q from here you can also observe the input/output of the carry function within Full adders.

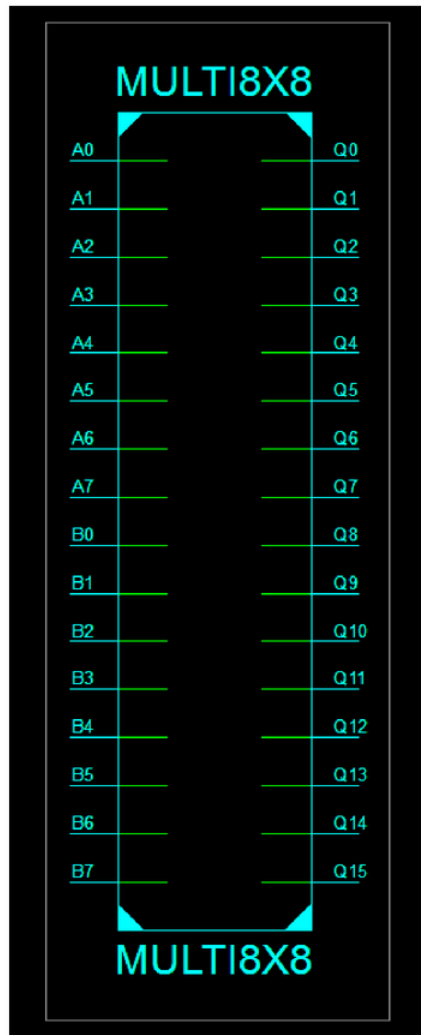


Multi8x8

The Multi8x8 bring together all previous coding together Building all the lines need for the 8x8 more multiple this will port map the all needed functions Line1 will be port-mapped first the linex will be port-mapped for the Line 2-8. This will be the final coding block need for 8x8 meaning that after this the test bench will show the 8x8 multiplication in action.

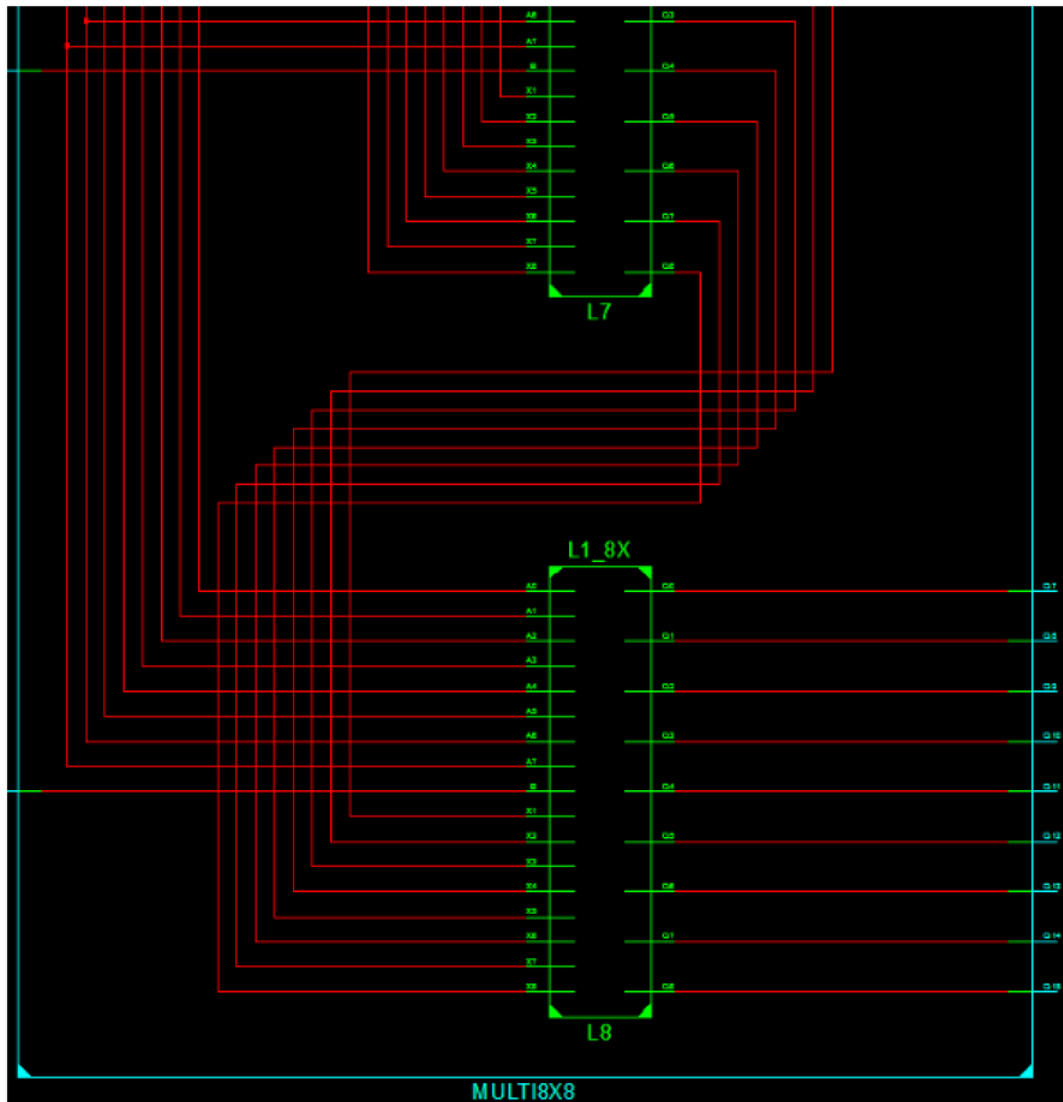
The Skmatic design

This schematic design shows all input A0-7, B0-B7 and output Q0-Q15 into the multi8x8 this is what the Microchip could look like if put into production. The Q output will show the Binary value of Mulpication



Multi8x8 Line 15

If you go deeper into the design black you can observe in Adder at work connect to the represent input and output. Line 8 of the design is shown here giving the outputs from Q7-Q15



VDHL code

As the 8x8 coding contain all Inputs/outputs and all need port mapping the amount of coding need has risen a lot. The below figure shows all need input and output for the multi 8x8 all set in standard logic.

```
entity MULTI8x8 is
  Port (
    A0 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
    A1 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
    A2 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
    A3 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
    A4 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
    A5 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
    A6 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
    A7 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC

    B0 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
    B1 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
    B2 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
    B3 : in  std_logic;    --SET INPUT STANDARD LOGIC
    B4 : in  std_logic;    --SET INPUT STANDARD LOGIC
    B5 : in  std_logic;    --SET INPUT STANDARD LOGIC
    B6 : in  std_logic;    --SET INPUT STANDARD LOGIC
    B7 : in  std_logic;    --SET INPUT STANDARD LOGIC

    Q0 : out  STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q1 : out  STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q2 : out  STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q3 : out  STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q4 : out  STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q5 : out  STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q6 : out  STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q7 : out  STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q8 : out  STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q9 : out  STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q10 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q11 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q12 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q13 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q14 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
    Q15 : out STD_LOGIC);   --SET OUTPUT STANDARD LOGIC
end MULTI8x8;
```

This show the components input/outputs for Line1 and line X calling these into the multiple files will give the ability to use them in the 8x8 multiplication

```

72
73 COMPONENT L1_8 is          -- SETTING LOGIC FOR LINE 1
74     Port ( A0 : in  STD_LOGIC;    --SET STANDARD LOGIC
75            A1 : in  STD_LOGIC;    --SET STANDARD LOGIC
76            A2 : in  STD_LOGIC;    --SET STANDARD LOGIC
77            A3 : in  STD_LOGIC;    --SET STANDARD LOGIC
78            A4 : in  STD_LOGIC;    --SET STANDARD LOGIC
79            A5 : in  STD_LOGIC;    --SET STANDARD LOGIC
80            A6 : in  STD_LOGIC;    --SET STANDARD LOGIC
81            A7 : in  STD_LOGIC;    --SET STANDARD LOGIC
82            B0 : in  STD_LOGIC;    --SET STANDARD LOGIC
83            Q0 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
84            Q1 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
85            Q2 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
86            Q3 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
87            Q4 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
88            Q5 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
89            Q6 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
90            Q7 : out STD_LOGIC);   --SET OUTPUT STANDARD LOGIC
91 END COMPONENT;
92
93 COMPONENT L1_8X is          -- SETTING LOGIC FOR LINE X
94     Port (
95
96            A0 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
97            A1 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
98            A2 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
99            A3 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
100           A4 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
101           A5 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
102           A6 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
103           A7 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
104           B  : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
105           Q0 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
106           Q1 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
107           Q2 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
108           Q3 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
109           Q4 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
110           Q5 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
111           Q6 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
112           Q7 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
113           Q8 : out STD_LOGIC;    --SET OUTPUT STANDARD LOGIC
114           X1 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
115           X2 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
116           X3 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
117           X4 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
118           X5 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
119           X6 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
120           X7 : in  STD_LOGIC;    --SET INPUT STANDARD LOGIC
121           X8 : in  STD_LOGIC);   --SET INPUT STANDARD LOGIC
122 END COMPONENT;

```

```

124
125     SIGNAL    Q1_1 : STD_LOGIC;      --SET STANDARD LOGIC LINE1
126     SIGNAL    Q1_2 : STD_LOGIC;      --SET STANDARD LOGIC LINE1
127     SIGNAL    Q1_3 : STD_LOGIC;      --SET STANDARD LOGIC LINE1
128     SIGNAL    Q1_4 : STD_LOGIC;      --SET STANDARD LOGIC LINE1
129     SIGNAL    Q1_5 : STD_LOGIC;      --SET STANDARD LOGIC LINE1
130     SIGNAL    Q1_6 : STD_LOGIC;      --SET STANDARD LOGIC LINE1
131     SIGNAL    Q1_7 : STD_LOGIC;      --SET STANDARD LOGIC LINE1
132
133
134
135     SIGNAL    Q2_1 : STD_LOGIC;      --SET STANDARD LOGIC LINE2
136     SIGNAL    Q2_2 : STD_LOGIC;      --SET STANDARD LOGIC LINE2
137     SIGNAL    Q2_3 : STD_LOGIC;      --SET STANDARD LOGIC LINE2
138     SIGNAL    Q2_4 : STD_LOGIC;      --SET STANDARD LOGIC LINE2
139     SIGNAL    Q2_5 : STD_LOGIC;      --SET STANDARD LOGIC LINE2
140     SIGNAL    Q2_6 : STD_LOGIC;      --SET STANDARD LOGIC LINE2
141     SIGNAL    Q2_7 : STD_LOGIC;      --SET STANDARD LOGIC LINE2
142     SIGNAL    Q2_8 : STD_LOGIC;      --SET STANDARD LOGIC LINE2
143
144
145     SIGNAL    Q3_1 : STD_LOGIC;      --SET STANDARD LOGIC LINE3
146     SIGNAL    Q3_2 : STD_LOGIC;      --SET STANDARD LOGIC LINE3
147     SIGNAL    Q3_3 : STD_LOGIC;      --SET STANDARD LOGIC LINE3
148     SIGNAL    Q3_4 : STD_LOGIC;      --SET STANDARD LOGIC LINE3
149     SIGNAL    Q3_5 : STD_LOGIC;      --SET STANDARD LOGIC LINE3
150     SIGNAL    Q3_6 : STD_LOGIC;      --SET STANDARD LOGIC LINE3
151     SIGNAL    Q3_7 : STD_LOGIC;      --SET STANDARD LOGIC LINE3
152     SIGNAL    Q3_8 : STD_LOGIC;      --SET STANDARD LOGIC LINE3
153
154
155     SIGNAL    Q4_1 : STD_LOGIC;      --SET STANDARD LOGIC LINE4
156     SIGNAL    Q4_2 : STD_LOGIC;      --SET STANDARD LOGIC LINE4
157     SIGNAL    Q4_3 : STD_LOGIC;      --SET STANDARD LOGIC LINE4
158     SIGNAL    Q4_4 : STD_LOGIC;      --SET STANDARD LOGIC LINE4
159     SIGNAL    Q4_5 : STD_LOGIC;      --SET STANDARD LOGIC LINE4
160     SIGNAL    Q4_6 : STD_LOGIC;      --SET STANDARD LOGIC LINE4
161     SIGNAL    Q4_7 : STD_LOGIC;      --SET STANDARD LOGIC LINE4
162     SIGNAL    Q4_8 : STD_LOGIC;      --SET STANDARD LOGIC LINE4
163
164
165     SIGNAL    Q5_1 : STD_LOGIC;      --SET STANDARD LOGIC LINES
166     SIGNAL    Q5_2 : STD_LOGIC;      --SET STANDARD LOGIC LINES
167     SIGNAL    Q5_3 : STD_LOGIC;      --SET STANDARD LOGIC LINES
168     SIGNAL    Q5_4 : STD_LOGIC;      --SET STANDARD LOGIC LINES
169     SIGNAL    Q5_5 : STD_LOGIC;      --SET STANDARD LOGIC LINES
170     SIGNAL    Q5_6 : STD_LOGIC;      --SET STANDARD LOGIC LINES
171     SIGNAL    Q5_7 : STD_LOGIC;      --SET STANDARD LOGIC LINES
172     SIGNAL    Q5_8 : STD_LOGIC;      --SET STANDARD LOGIC LINES
173

```

Setting all output to stand logic will make sure that there are no issues when calling needed values.

This shows the final port mapping for the 8x8 multiple port mapping for this device is done by stating inputs on the left with outputs of the right. Q0-Q15 will give all the final values for this 8x8.

```

195
196 begin
197
198     L1 : L1_8      --PORT MAP LINE 1
199     PORT MAP ( A0,A1,A2,A3,A4,A5,A6,A7,B0,Q0,Q1_1,Q1_2,Q1_3,Q1_4,Q1_5,Q1_6,Q1_7);
200
201     L2 : L1_8X     --PORT MAP LINE 2
202     PORT MAP ( A0,A1,A2,A3,A4,A5,A6,A7,B1,Q1,Q2_1,Q2_2,Q2_3,Q2_4,Q2_5,Q2_6,Q2_7,Q2_8,Q1_1,Q1_2,Q1_3,Q1_4,Q1_5,Q1_6,Q1_7,'0');
203
204     L3 : L1_8X     --PORT MAP LINE 3
205     PORT MAP ( A0,A1,A2,A3,A4,A5,A6,A7,B2,Q2,Q3_1,Q3_2,Q3_3,Q3_4,Q3_5,Q3_6,Q3_7,Q3_8,Q2_1,Q2_2,Q2_3,Q2_4,Q2_5,Q2_6,Q2_7,Q2_8);
206
207     L4 : L1_8X     --PORT MAP LINE 4
208     PORT MAP ( A0,A1,A2,A3,A4,A5,A6,A7,B3,Q3,Q4_1,Q4_2,Q4_3,Q4_4,Q4_5,Q4_6,Q4_7,Q4_8,Q3_1,Q3_2,Q3_3,Q3_4,Q3_5,Q3_6,Q3_7,Q3_8);
209
210     L5 : L1_8X     --PORT MAP LINE 5
211     PORT MAP ( A0,A1,A2,A3,A4,A5,A6,A7,B4,Q4,Q5_1,Q5_2,Q5_3,Q5_4,Q5_5,Q5_6,Q5_7,Q5_8,Q4_1,Q4_2,Q4_3,Q4_4,Q4_5,Q4_6,Q4_7,Q4_8);
212
213     L6 : L1_8X     --PORT MAP LINE 6
214     PORT MAP ( A0,A1,A2,A3,A4,A5,A6,A7,B5,Q5,Q6_1,Q6_2,Q6_3,Q6_4,Q6_5,Q6_6,Q6_7,Q6_8,Q5_1,Q5_2,Q5_3,Q5_4,Q5_5,Q5_6,Q5_7,Q5_8);
215
216     L7 : L1_8X     --PORT MAP LINE 7
217     PORT MAP ( A0,A1,A2,A3,A4,A5,A6,A7,B6,Q6,Q7_1,Q7_2,Q7_3,Q7_4,Q7_5,Q7_6,Q7_7,Q7_8,Q6_1,Q6_2,Q6_3,Q6_4,Q6_5,Q6_6,Q6_7,Q6_8);
218
219     L8 : L1_8X     --PORT MAP LINE 8
220     PORT MAP ( A0,A1,A2,A3,A4,A5,A6,A7,B7,Q7,Q8,Q9,Q10,Q11,Q12,Q13,Q14,Q15,Q7_1,Q7_2,Q7_3,Q7_4,Q7_5,Q7_6,Q7_7,Q7_8);
221
222
223 end Behavioral;
224

```

Test bench Code

Test bench Code

This shows the final coding for the VHDL test bench there will be a need to observe the output signals

Part1 VHDL Multi test bench

```
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;
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;
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
```

Part2 VHDL Multi test bench

```
80  --Inputs
81  signal A_0 : std_logic := '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;
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
```

Part3 VHDL Multi test bench

```
80  --Inputs
81  signal A_0 : std_logic := '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;
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
```

Part 4 VHDL Multi test bench

```
163 stim_proc0: process
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;
```

Part5 VHDL Multi test bench

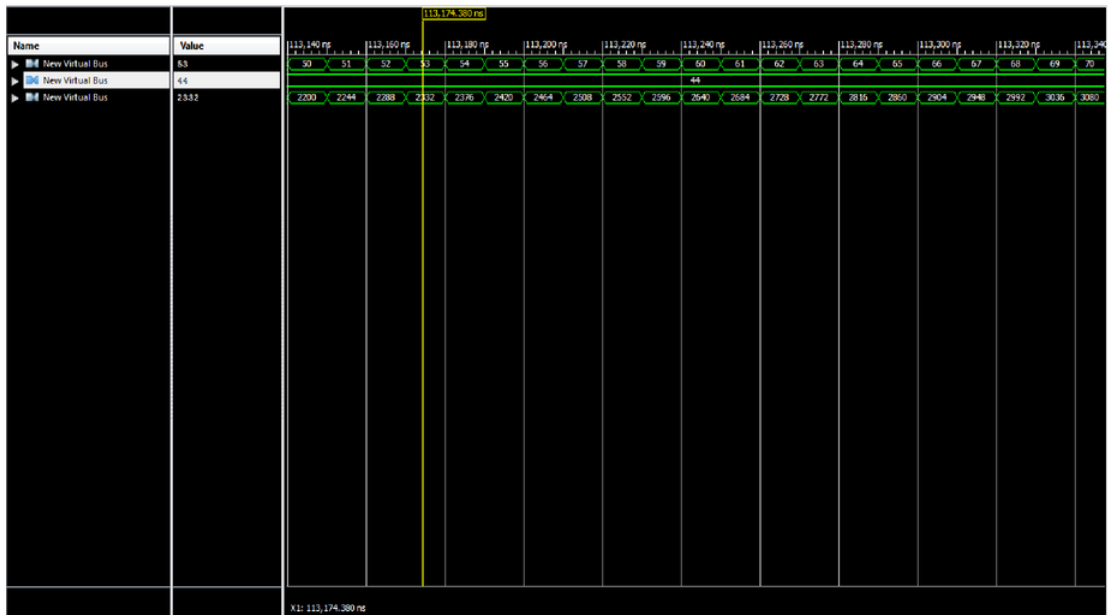
```
212      -- PORT B
213      stim_proc10: process
214      begin
215          wait for 2560 ns;
216          B_0 <= NOT B_0;
217      end process;
218
219      stim_proc11: process
220      begin
221          wait for 5120 ns;
222          B_1 <= NOT B_1;
223      end process;
224
225      stim_proc12: 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;
```

Multi Signals

This shows the multi 8x8 functioning correctly in the below diagram we have taken the multiple to it limits formula $A \times B$ in decimal $255 \times 255 = 65025$, Binary $11111111 \times 11111111 = 1111111000000001$



From the above test bench, this report can confirm it has successfully created an 8x8 multiplier being able to multiply 8 binary numbers by 8 binary numbers.



This test bench was taken a bit more in the middle of the results to show that all areas are covered by this 8x8 the above shows $A \times B = Q$ used with the values $53 \times 44 = 2332$ This again shows that the 8x8 is working correctly covering all areas

Observation

From all the rest bench operation this report can observe the deployment of the 8x8 multiplier builds on the Braun architecture. Being able to simulate and brake down the components into there Gates shows how the Braun architecture is constructed using Gates and adders. This project has successfully simulated the 8x8 multiplex as shown in the test bench for the multi8x8 segment.



Here show one of the test bench signals from the final multi8x8 Here you can see the Binary multiplication at work You can see the (A input group X B input group = Q) or in decimal $242 \times 20 = 2840$.

VHDL code/models

All VDHL code has been animated to show each function building a better understanding of how VDHL code works. Developing all Port mapping, Gate creation and I/O declarations the VDHL code is effective and easy to follow.

Analysis

The project has created an 8x8 multiplication of signed 2's complement numbers as asked for in Requirements. Developing the project has its difficult with a low knowledge of VHDL, to begin with, but as developing through this project that understanding has grown to give an outline of how VHDL functions.

From the begin of this project, this report has made sure to keep all values in standard logic and not use vectors this was because understanding the Flow of the 8x8 multiplex was vital and vector did not seem like the best means for a better understanding of this project.

Not using vectors later on in the assignment paid off as there were compactions with the software used when using vectors this was avoided.

Each sector shows the Test bench, VHDL Code and Signals each of these sectors creates the building blocks of building and testing coding. This was very useful to confirm that coding was correct.

Each sector was tested using test bench that was in build to confirm there signal using Xilinx ISE.

Building a diagram of the 8x8 multiplier on Draw.io give the ability to analyse the structure of the 8x8 and be able to transfers to coding.

Conclusion

From the begin of the project, all aims and goals have been meet created a 8x8 multiplication of signed 2's complement numbers this was achieved by using advanced Xilinx software that has been designing to be used in correlation with VHDL(VHSIC Hardware Description Language) these together have been all the tools needed for this report. Building a diagram of 8x8 architecture gave a better understanding of how this architecture would flow in VHDL code. When building the 8x8 in Xilinx software it was best to break it into part first starting with the half adder and full adder these will be the basic building block of this report. After creating your adders then advancing on to your first line of this would be made up of And Gates and only be called once in the first line. Creating line x would fill in the gaps from line 2-8 this is as the same structure flows thought. This structure would consist of 7 full adders and 1 half adder.

Building the coding the final file the mulit8x8 file brought all previse coding together allowing for the creation of the final product this section would port map each need line for all there need inputs and outputs from the test bench of this final file the observation and analyse of the final signal showed that a successful 8x8 multiplier was created. The final multiplier was able to all need calculation being able to multiply 255 by 255 this is the most an 8x8 multiplier could create.

This project has created a larger understanding of VHSIC Hardware Description Language and how to use this in conjunction with Xilinx software to build an 8x8 multiplication.

References

- [1] “Design of Braun Multiplier with Kogge Stone Adder & Its Implementation on FPGA.” [Online]. Available: <https://www.ijser.org/paper/Design-of-Braun-Multiplier-with-Kogge-Stone-Adder-Its-Implementation-on-FPGA.html>. [Accessed: 11-May-2018].
- [2] A. Davies and Y. Fung, “Interfacing a hardware multiplier to a general-purpose microprocessor,” *Microprocessors*, vol. 1, no. 7, pp. 425–432, Oct. 1977.
- [3] A. Nelapati, L. W. Jesima, and V. Bagyaveereswaran, “Comparative Study of High-performance Braun’s Multiplier using FPGAs,” *IOSR J. Electron. Commun. Eng.*, vol. 1, no. 4, pp. 2278–2834.
- [4] G. Balachandran, N. Kandasamysuryaprakash, and T. N. Devaprakash, “Design of Low power multipliers with Braun architecture using column bypassing multipliers.”
- [5] “THE MULTIPLIER.”
- [6] “PRINCIPLES OF HDL.”
- [7] “VHDL Tutorial - Introduction to VHDL for beginners.” [Online]. Available: <https://www.nandland.com/vhdl/tutorials/tutorial-introduction-to-VHDL-for-beginners.html>. [Accessed: 11-May-2018].
- [8] *1076-1987 – IEEE Standard VHDL Language Reference Manual*. 1988.

FINAL GRADE

GENERAL COMMENTS

Instructor

60/100

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

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

GRADING FORM: SMU GENERIC_GJ

RYAN O'SULLIVAN

59.50

RESEARCH-INFORMED LITERATURE



Very limited research with no other multipliers designs, could have been vastly improved more types and a comparison table. 45%

4.5

KNOWLEDGE AND UNDERSTANDING



You have shown competency and understanding with your VHDL code. Your code is readable and remarked. 66%

19.8

ANALYSIS



You have demonstrated your analysis of the design at each stage and commented on each section, with brief explanations of operation. 61%

12.2

PRACTICAL APPLICATION AND DEP.



You have simulation results, demonstrating the operation of the circuit but more images and discussion would have been welcome. Not every section of the design has been shown to have been tested before reuse in the next section. 56%

16.8

SKILLS FOR PROFESSIONAL PRAC.



The report is well presented with good diagrams and illustrations. A bibliography and small selection of references is included. 62%

6.2