



SDSD LAB PROJECT

Topic:-Vedic Multiplier(4 and 8 Bit)

PRESENTED By:-

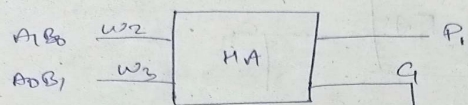
Narendra Kumar Nehra

4 Bit Vedic Multiplier

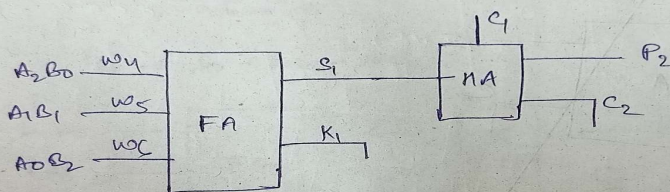
				A_3	A_2	A_1	A_0
				B_3	B_2	B_1	B_0
			$A_3 B_0$	$A_2 B_0$	$A_1 B_0$	$A_0 B_0$	
		$A_3 B_1$	$A_2 B_1$	$A_1 B_1$	$A_0 B_1$		X
	$A_3 B_2$	$A_2 B_2$	$A_1 B_2$	$A_0 B_2$		X	
$A_3 B_3$	$A_2 B_3$	$A_1 B_3$	$A_0 B_3$		X		
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0
							$P_0 = A_0 B_0$
							$P_1 = A_1 B_0 + A_0 B_1$
							$P_2 = A_2 B_0 + A_1 B_1 + A_0 B_2$
							$P_3 = A_3 B_0 + A_2 B_1 + A_1 B_2 + A_0 B_3$
							$P_4 = A_3 B_1 + A_2 B_2 + A_1 B_3$
							$P_5 = A_3 B_2 + A_2 B_3$
							$P_6 = A_3 B_3$, $P_7 = \text{carry from } P_6$

$$A_0 B_0 \xrightarrow{w_1} P_0$$

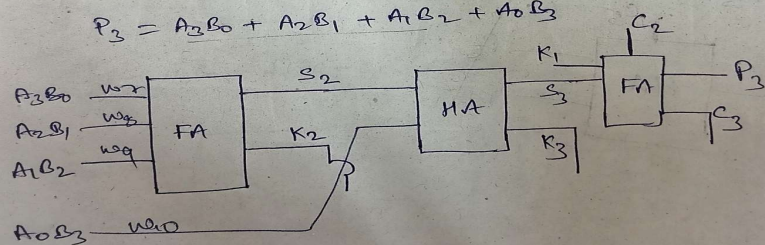
$$P_1 = A_1 B_0 + A_0 B_1$$



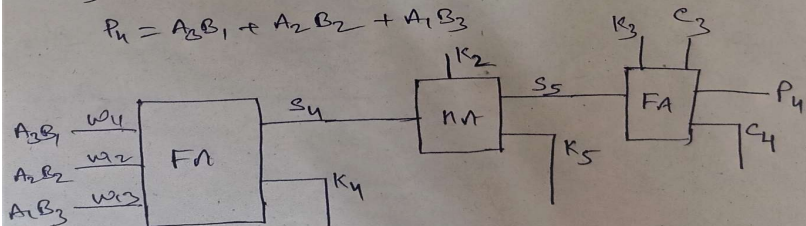
$$P_2 = A_2 B_0 + A_1 B_1 + A_0 B_2$$



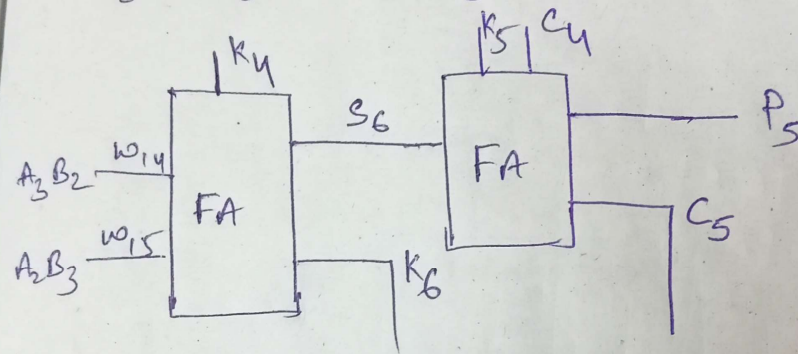
$$P_3 = A_3 B_0 + A_2 B_1 + A_1 B_2 + A_0 B_3$$



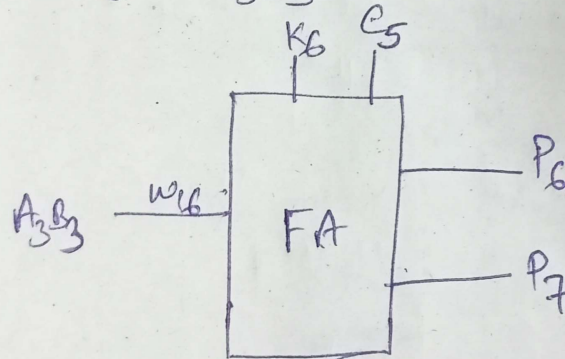
$$P_4 = A_3 B_1 + A_2 B_2 + A_1 B_3$$



$$P_5 = A_3 B_2 + A_2 B_3$$



$$P_6 = A_3 B_3$$



Code for 4 Bit

```

12
13 module fourbitmultiplier(p,a,b);
14 input [3:0]a;
15 input [3:0]b;
16 output [7:0]p;
17 wire [7:0]p;
18 wire c1,c2,c3,c4,c5,k1,k2,k3,k4,k5,k6,s1,s2,s3,s4,s5,s6;
19 wire w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14,w15,w16;
20 and X1(p[0],a[0],b[0]);
21
22 and X2(w2,a[1],b[0]);
23 and X3(w3,a[0],b[1]);
24 halfadder HA1(p[1],c1,w2,w3);
25
26 and A4(w4,a[2],b[0]);
27 and A5(w5,a[1],b[1]);
28 and A6(w6,a[0],b[2]);
29 fulladder FA1(s1,k1,w4,w5,w6);
30 halfadder HA2(p[2],c2,s1,c1);
31
32 and A7(w7,a[3],b[0]);
33 and A8(w8,a[2],b[1]);
34 and A9(w9,a[1],b[2]);
35 fulladder FA2(s2,k2,w7,w8,w9);
36 and A10(w10,a[0],b[3]);
37 halfadder HA3(s3,k3,s2,w10);
38 fulladder FA3(p[3],c3,s3,k1,c2);
39
40 and A11(w11,a[3],b[1]);
41 and A12(w12,a[2],b[2]);
42 and A13(w13,a[1],b[3]);
43 fulladder FA4(s4,k4,w11,w12,w13);
44 halfadder HA4(s5,k5,s4,k2);
45 fulladder FA5(p[4],c4,s5,k3,c3);
46
47 and A14(w14,a[3],b[2]);
48 and A15(w15,a[2],b[3]);
49 fulladder FA6(s6,k6,w14,w15,k4);
50 fulladder FA7(p[5],c5,s6,k5,c4);
51
52 and A16(w16,a[3],b[3]);
53 fulladder FA8(p[6],p[7],w16,k6,c5);
54 endmodule
55

```



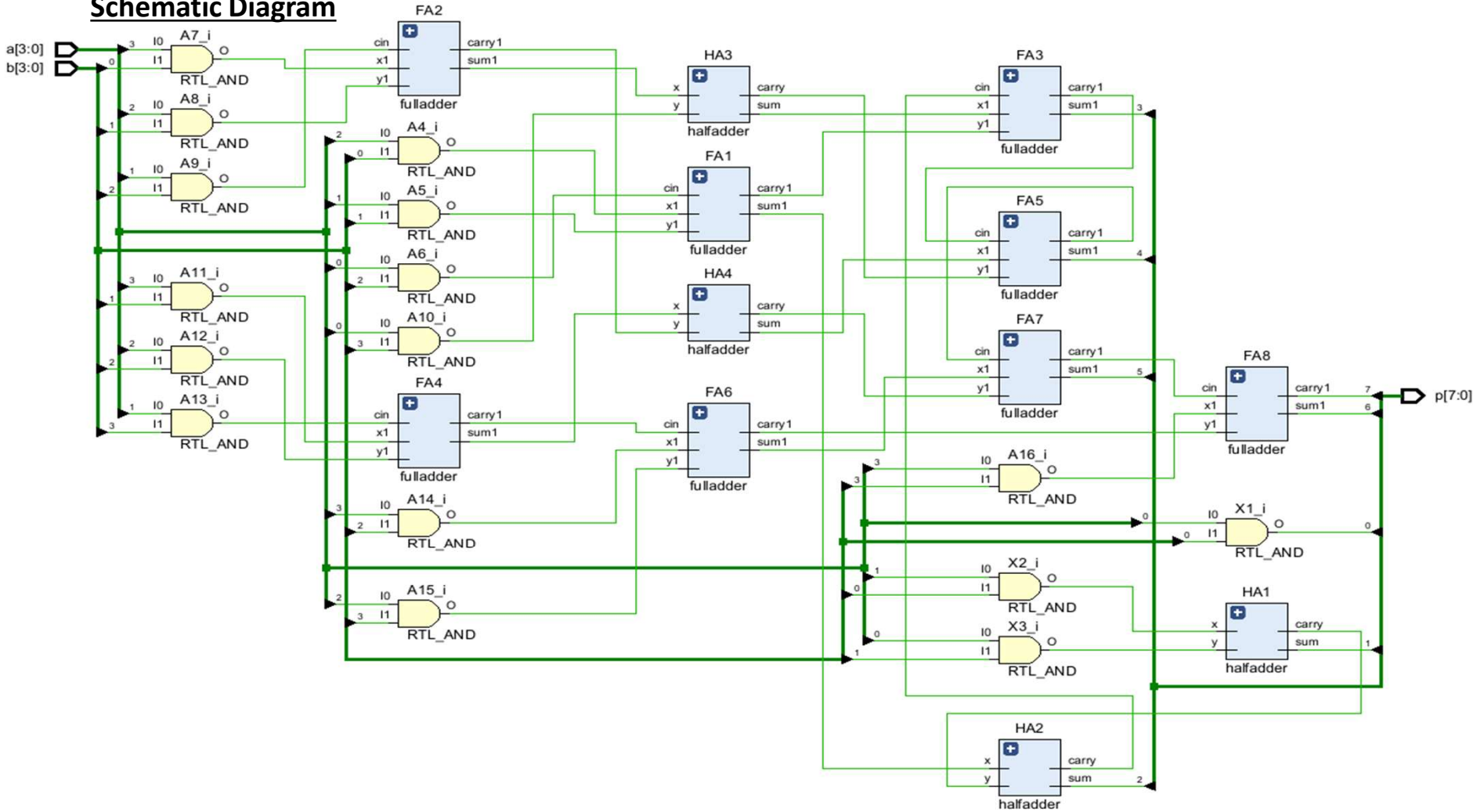
```

51 |
52 | ○ and A16(w16,a[3],b[3]);
53 | fulladder FA8(p[6],p[7],w16,k6,c5);
54 | ⊖ endmodule
55 |
56 |
57 | ⊖ module halfadder(sum,carry,x,y);
58 |   input x,y;
59 |   output sum,carry;
60 |   wire sum,carry;
61 |   ○ xor X1(sum,x,y);
62 |   ○ and Y1(carry,x,y);
63 | ⊖ endmodule
64 |
65 | ⊖ module fulladder(sum1,carry1,x1,y1,cin);
66 |   input x1,y1,cin;
67 |   output sum1,carry1;
68 |   wire k1,k2,k3;
69 |   ○ xor X1(k1,x1,y1);
70 |   ○ xor X2(sum1,cin,k1);
71 |   ○ and X3(k2,cin,k1);
72 |   ○ and X4(k3,x1,y1);
73 |   ○ or X5(carry1,k3,k2);
74 | ⊖ endmodule
-- |

```

Schematic Diagram

The schematic diagram illustrates a 4-bit ripple-carry adder. It takes two 4-bit inputs, `a[3:0]` and `b[3:0]`, and produces a 4-bit output `p[7:0]`. The circuit is composed of several full adders (FA2, FA3, FA4, FA5, FA6, FA7, FA8) and half adders (HA3, HA4, HA1, HA2). The inputs are connected to the adders through a network of RTL_AND gates. The carry chain starts with a carry-in of 0 and propagates through the adders to produce the final 4-bit sum `p[7:0]`.



Test Bench

Result

```
# run 1000ns
```

```
0a= 1110,b= 0111,p= 98
5a= 1000,b= 1000,p= 64
10a= 1111,b= 1111,p= 225
15a= 1110,b= 1110,p= 196
20a= 0111,b= 1101,p= 91
25a= 1011,b= 1101,p= 143
```

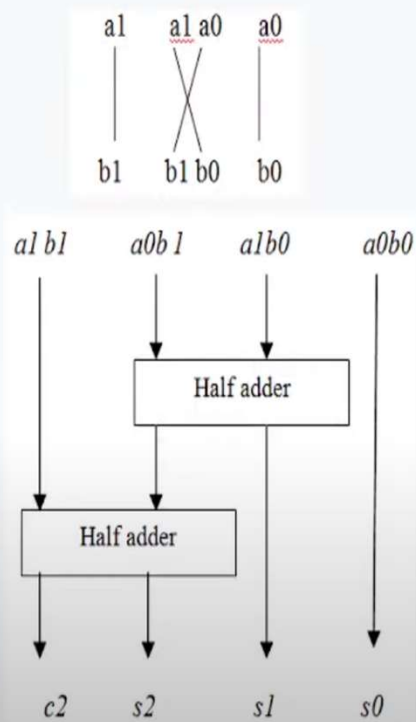
Waveform

Name	Value	0.000 ns	10.000 ns	20.000 ns	30.000 ns	40.000 ns	50.000 ns	60.000 ns	70.000 ns	80.000 ns	90.000 ns
> p[0:7]	8f	62	40	e1	c4	5b		8f			
> a[0:3]	b	e	8	f	e	7		b			
> b[0:3]	d	7	8	f	e			d			

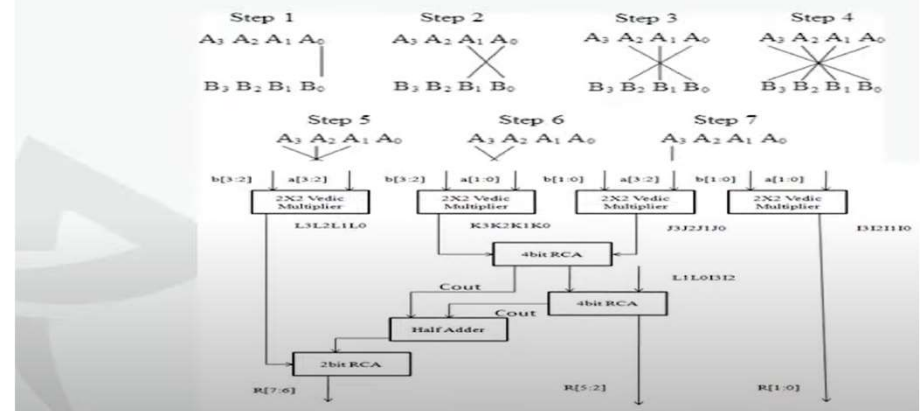
									y ₇	y ₆	y ₅	y ₄	y ₃	y ₂	y ₁	y ₀
									x ₇	x ₆	x ₅	x ₄	x ₃	x ₂	x ₁	x ₀
									p ₇₀	p ₆₀	p ₅₀	p ₄₀	p ₃₀	p ₂₀	p ₁₀	p ₀₀
								p ₇₁	p ₆₁	p ₅₁	p ₄₁	p ₃₁	p ₂₁	p ₁₁	p ₀₁	
							p ₇₂	p ₆₂	p ₅₂	p ₄₂	p ₃₂	p ₂₂	p ₁₂	p ₀₂		
						p ₇₃	p ₆₃	p ₅₃	p ₄₃	p ₃₃	p ₂₃	p ₁₃	p ₀₃			
					p ₇₄	p ₆₄	p ₅₄	p ₄₄	p ₃₄	p ₂₄	p ₁₄	p ₀₄				
				p ₇₅	p ₆₅	p ₅₅	p ₄₅	p ₃₅	p ₂₅	p ₁₅	p ₀₅					
			p ₇₆	p ₆₆	p ₅₆	p ₄₆	p ₃₆	p ₂₆	p ₁₆	p ₀₆						
		p ₇₇	p ₆₇	p ₅₇	p ₄₇	p ₃₇	p ₂₇	p ₁₇	p ₀₇							
s ₁₅	s ₁₄	s ₁₃	s ₁₂	s ₁₁	s ₁₀	s ₉	s ₈	s ₇	s ₆	s ₅	s ₄	s ₃	s ₂	s ₁	s ₀	

8 Bit Vedic Multiplier

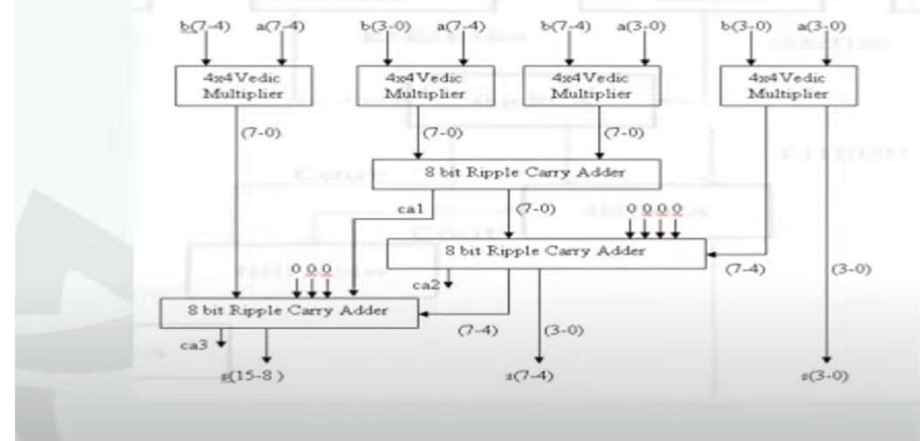
2-bit Vedic Multiplier



4-bit Vedic Multiplier



8-bit Vedic Multiplier



Advantages

1. consumes 66% less area.
2. 76.1% less power.
3. 60% less delay

Code for 8-bit

```
13 module vedic8bit(output [15:0] out,input [7:0] in1,in2);
14     wire [7:0] w1,w2,w3,w4,s1,s2;
15     wire c1,c2,c3;
16     wire [7:0] W1,W2;
17
18     VM4bit VM4BIT_1(w1,in1[3:0],in2[3:0]);
19     VM4bit VM4BIT_2(w2,in1[3:0],in2[7:4]);
20     VM4bit VM4BIT_3(w3,in1[7:4],in2[3:0]);
21     VM4bit VM4BIT_4(w4,in1[7:4],in2[7:4]);
22
23     buf(s1[0],w1[4]);
24     buf(s1[1],w1[5]);
25     buf(s1[2],w1[6]);
26     buf(s1[3],w1[7]);
27     buf(s1[4],0);
28     buf(s1[5],0);
29     buf(s1[6],0);
30     buf(s1[7],0);
31
32
33     ADDER8bit A8_1( c1,W1, w2,w3);
34     ADDER8bit A8_2( c2,W2,W1,s1);
35
36     or o_1(c3,c1,c2);
37
38     buf(s2[0],W2[4]);
39     buf(s2[1],W2[5]);
40     buf(s2[2],W2[6]);
41     buf(s2[3],W2[7]);
42     buf(s2[4],c3);
43     buf(s2[5],0);
44     buf(s2[6],0);
45     buf(s2[7],0);
46
47
48
49
50     assign out[0] = w1[0];
51     assign out[1] = w1[1];
52     assign out[2] = w1[2];
53     assign out[3] = w1[3];
54     assign out[4] = W2[0];
55     assign out[5] = W2[1];
56     assign out[6] = W2[2];
```

```
53     assign out[3] = w1[3];
54     assign out[4] = W2[0];
55     assign out[5] = W2[1];
56     assign out[6] = W2[2];
57     assign out[7] = W2[3];
58
59
60     ADDER8bit A8_3(extra,out [15:8], w4,s2);
61
62
63 endmodule
64
65
66 module VM4bit(output [7:0] out,input [3:0] in1,in2);
67     wire [3:0] w1,w2,w3,w4,s1,s2;
68     wire c1,c2,c3;
69     wire [3:0] W1,W2;
70
71     VM2bit VM2BIT_1(w1,in1[1:0],in2[1:0]);
72     VM2bit VM2BIT_2(w2,in1[1:0],in2[3:2]);
73     VM2bit VM2BIT_3(w3,in1[3:2],in2[1:0]);
74     VM2bit VM2BIT_4(w4,in1[3:2],in2[3:2]);
75
76     buf(s1[0],w1[2]);
77     buf(s1[1],w1[3]);
78     buf(s1[2],0);
79     buf(s1[3],0);
80
81
82     ADDER4bit A4_1( c1,W1, w2,w3);
83     ADDER4bit A4_2( c2,W2,W1,s1);
84
85     or o_1(c3,c1,c2);
86
87     buf(s2[0],W2[2]);
88     buf(s2[1],W2[3]);
89     buf(s2[2],c3);
90     buf(s2[3],0);
91
92
93
94
95     assign out[0] = w1[0];
96     assign out[1] = w1[1];
```

```

94
95 assign out[0] = w1[0];
96 assign out[1] = w1[1];
97 assign out[2] = w2[0];
98 assign out[3] = w2[1];
99
100 ADDER4bit A4_3(extra,out [7:4], w4,s2);
101
102
103 endmodule
104
105 module VM2bit(output [3:0] out, input [1:0] A,B);
106
107 wire [3:0] s;
108
109 and a1(s[0],A[0],B[0]);
110 and a2(s[1],A[0],B[1]);
111 and a3(s[2],A[1],B[0]);
112 and a4(s[3],A[1],B[1]);
113
114 assign out[0] = s[0];
115 halfadder H_1(out[1],C,s[1],s[2]);
116 halfadder H_2(out[2],out[3],s[3],C);
117
118 endmodule
119
120 module ADDER8bit(C,out,A,B );
121 output [7:0] out;
122 output C;
123
124
125
126 input [7:0] A,B;
127 halfadder H_1(out[0],c1,A[0],B[0]);
128 fulladder F_1(out[1],c2,A[1],B[1],c1);
129 fulladder F_2(out[2],c3,A[2],B[2],c2);
130 fulladder F_3(out[3],c4,A[3],B[3],c3);
131 fulladder F_4(out[4],c5,A[4],B[4],c4);
132 fulladder F_5(out[5],c6,A[5],B[5],c5);
133 fulladder F_6(out[6],c7,A[6],B[6],c6);
134 fulladder F_7(out[7],C,A[7],B[7],c7);
135
136 endmodule
137

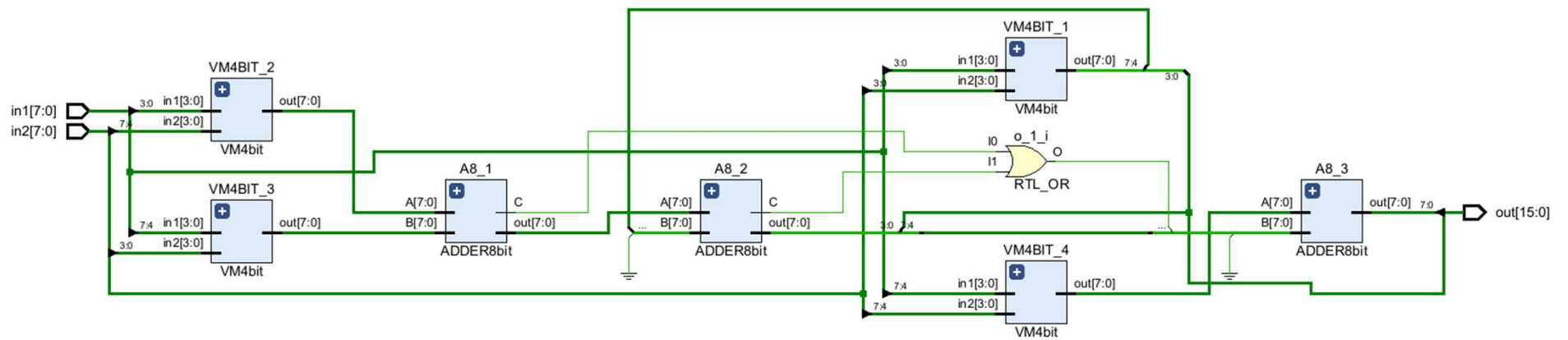
```

```

136 endmodule
137
138
139 module ADDER4bit(C,out,A,B );
140 output [3:0] out;
141 output C;
142
143
144 input [3:0] A,B;
145 halfadder H_1(out[0],c1,A[0],B[0]);
146 fulladder F_1(out[1],c2,A[1],B[1],c1);
147 fulladder F_2(out[2],c3,A[2],B[2],c2);
148 fulladder F_3(out[3],C,A[3],B[3],c3);
149 endmodule
150
151
152 module halfadder(sum, carry ,A,B);
153 output sum,carry;
154 input A,B;
155
156 assign sum = A^B;
157 assign carry = A&B;
158 endmodule
159
160 module fulladder(sum, carry ,A,B, cin );
161 output sum,carry;
162 input A,B,cin;
163
164
165 assign sum = A^B^cin;
166 assign carry = A&B|B&cin|A&cin;
167
168 endmodule
169

```

Schematic Diagram



Test Bench

Result

```
13 :  
14 module vedic8bit_tb();  
15 :  
16 wire [0:15]out;  
17 reg [0:7]in1,in2;  
18 vedic8bit Multiplier4_bit(out,in1,in2);  
19 initial  
20 begin  
21 in1=8'b0001110; in2=8'b00001110;  
22 #5 in1=8'b10001000; in2=8'b11000000;  
23 #5 in1=8'b10001110; in2=8'b11000011;  
24 end  
25 initial $monitor($time,"in1= %d,in2= %d,out= %d", in1, in2, out);  
26 initial #100 $stop;  
27 endmodule
```

run 1000ns

0in1= 14,in2= 14,out= 196

5in1= 136,in2= 192,out= 26112

10in1= 142,in2= 195,out= 27690

THANK YOU