Vedic Multiplier is a key tool in rapidly growing technology especially in the immense domain of image processing, Digital Signal Processing, real-time signal. Multipliers are important block in digital systems and play a critical role in digital designs.

Advantages

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

The Vedic multiplier is a technique used in ancient Indian mathematics, specifically within Vedic mathematics, for multiplying two numbers. It involves breaking down the numbers, applying mental calculation using specific sutras, and combining the results to obtain the final product.

How it is different from normal multiplier?

VEDIC multiplier is a good alternative to the other fast multiplicative algorithms. VEDIC multipliers reduces hardware as well as the delay compared to other algorithms.

1. **Methodology:-**

The Vedic multiplier employs specific sutras from Vedic mathematics to simplify and speed up the multiplication process. It focus on mental calculations and breaking down the numbers into smaller components, whereas the normal multiplier follows a step by step algorithm approach.

2. **Speed and Efficiency:-**

The Vedic multiplier is known for its speed and efficiency in mental calculations. It allows for quicker computations compared to the conventional long multiplication method, particularly for certain types of numbers.

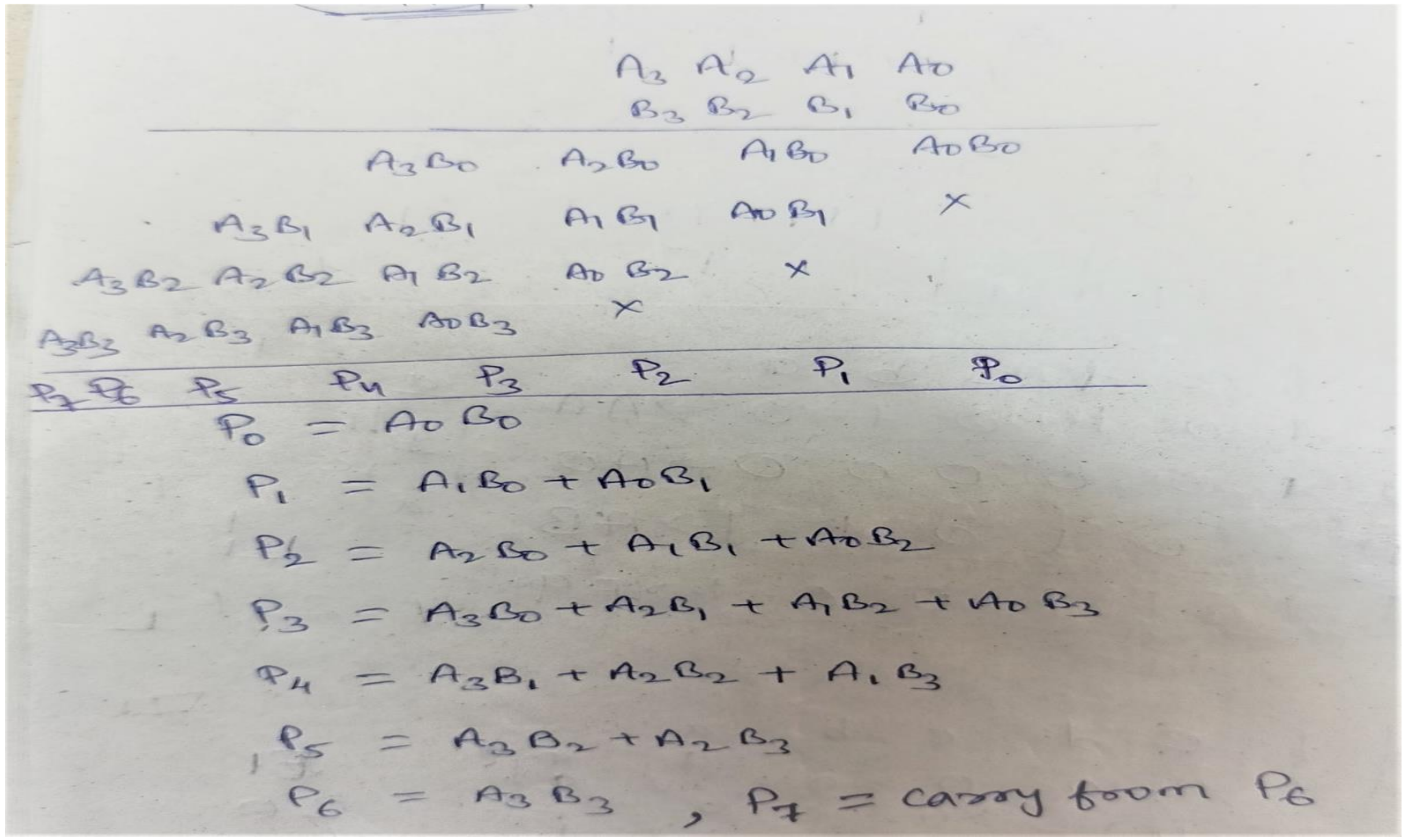
3. **Flexibility:-**

The Vedic multiplier offers more flexibility in terms of the numbers it can handle. It can be applied to various types of numbers, including large numbers, decimals, and fractions, making it a versatile technique.

4. **Educational Focus:-**

The Vedic multiplier is often taught as part of vedic mathematics courses or mental calculation techniques. It aims to enhance mathematical skills, mental ability, and overall number sense, while the normal multiplier is typically taught as a standard multiplication algorithm in traditional mathematics education.

4 Bit Vedic Multiplier



			A_3	A_2	A_1	A_0
			B_3	B_2	B_1	B_0
			A_3B_0	A_2B_0	A_1B_0	A_0B_0
		A_3B_1	A_2B_1	A_1B_1	A_0B_1	×
	A_3B_2	A_2B_2	A_1B_2	A_0B_2	×	
A_3B_3	A_2B_3	A_1B_3	A_0B_3	×		
P_7 P_6	P_5	P_4	P_3	P_2	P_1	P_0

$P_0 = A_0B_0$

$P_1 = A_1B_0 + A_0B_1$

$P_2 = A_2B_0 + A_1B_1 + A_0B_2$

$P_3 = A_3B_0 + A_2B_1 + A_1B_2 + A_0B_3$

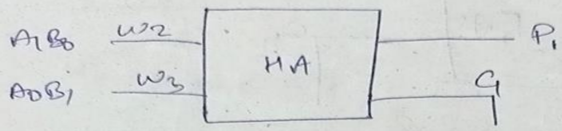
$P_4 = A_3B_1 + A_2B_2 + A_1B_3$

$P_5 = A_3B_2 + A_2B_3$

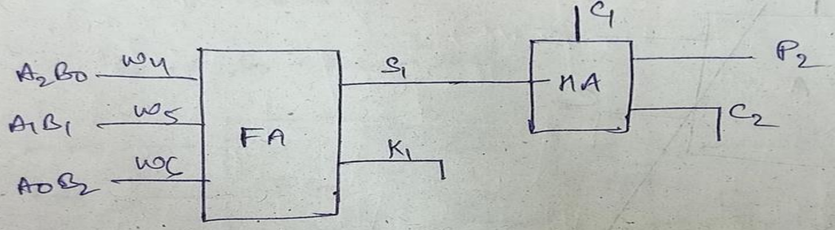
$P_6 = A_3B_3$, P_7 = carry from P_6

$A_0B_0 \xrightarrow{w_1} P_0$

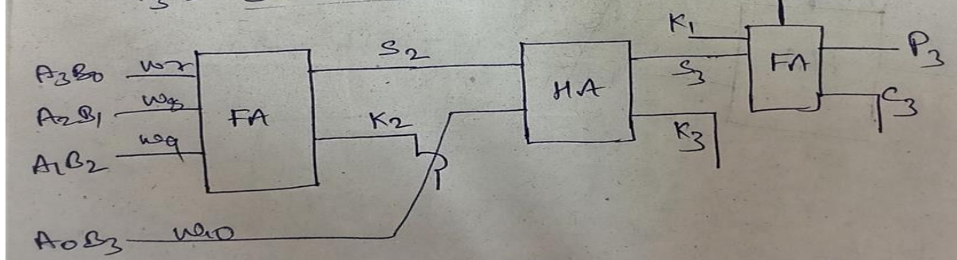
$P_1 = A_1B_0 + A_0B_1$



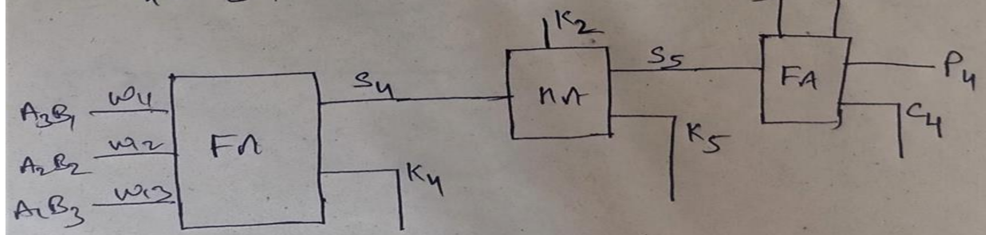
$P_2 = A_2B_0 + A_1B_1 + A_0B_2$



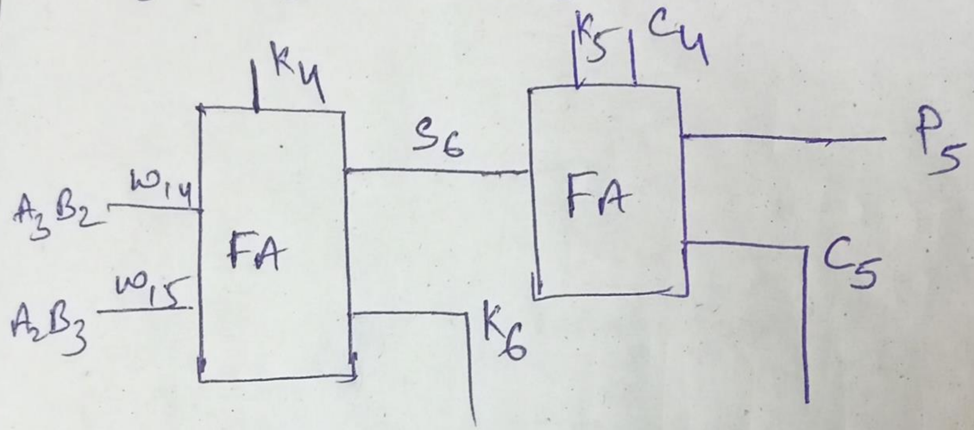
$P_3 = A_3B_0 + A_2B_1 + A_1B_2 + A_0B_3$



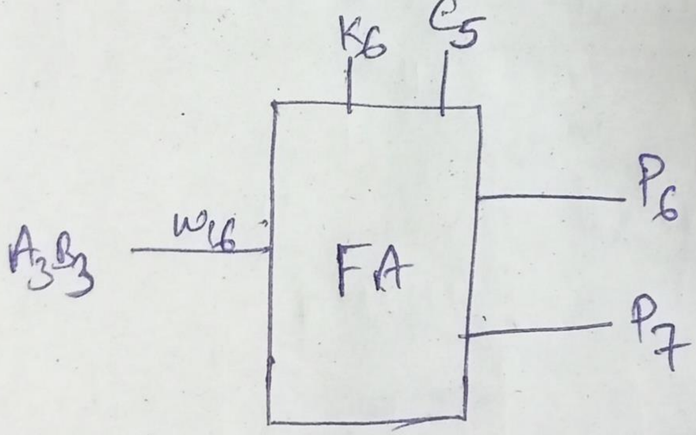
$P_4 = A_3B_1 + A_2B_2 + A_1B_3$



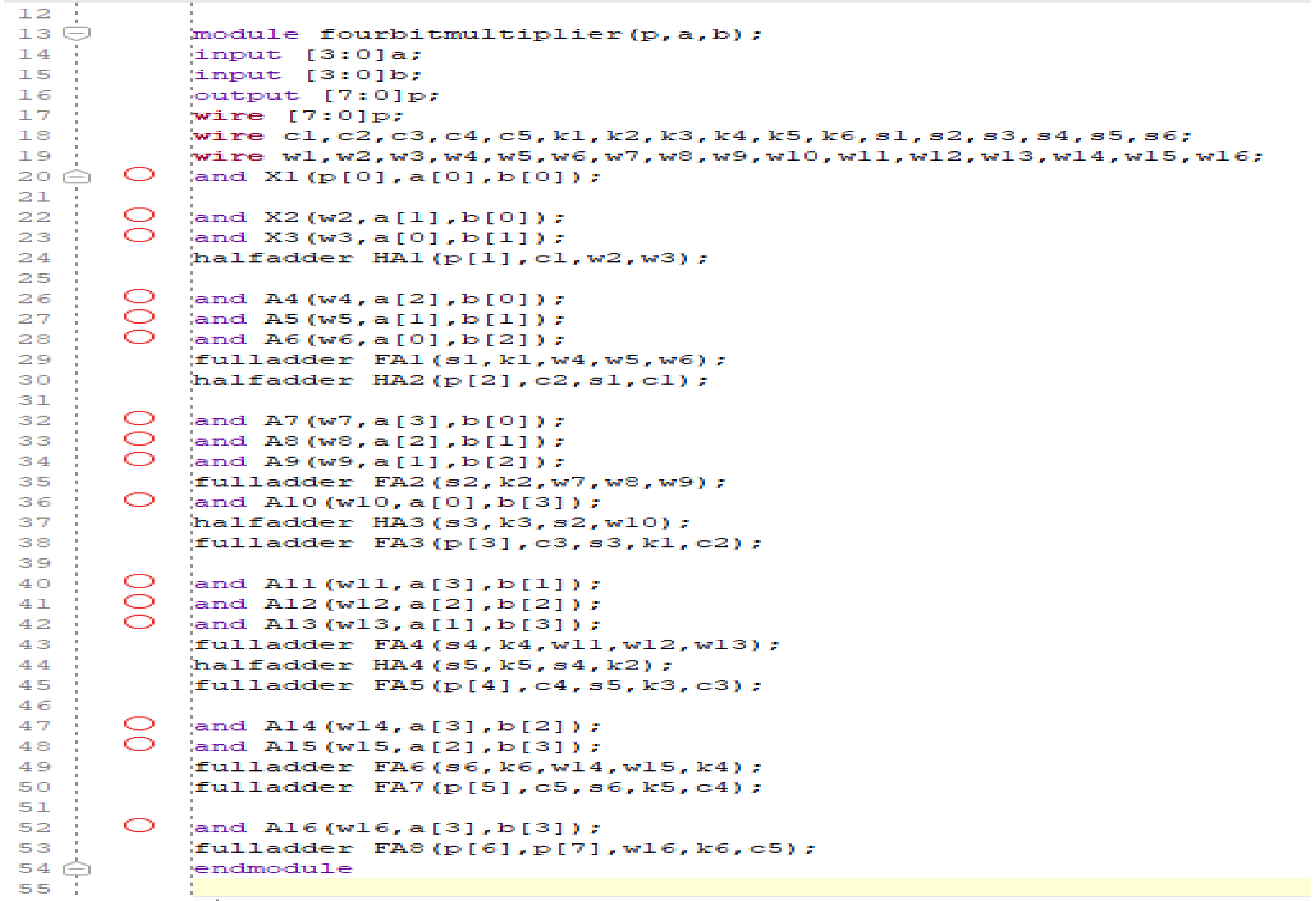
$P_5 = A_3B_2 + A_2B_3$



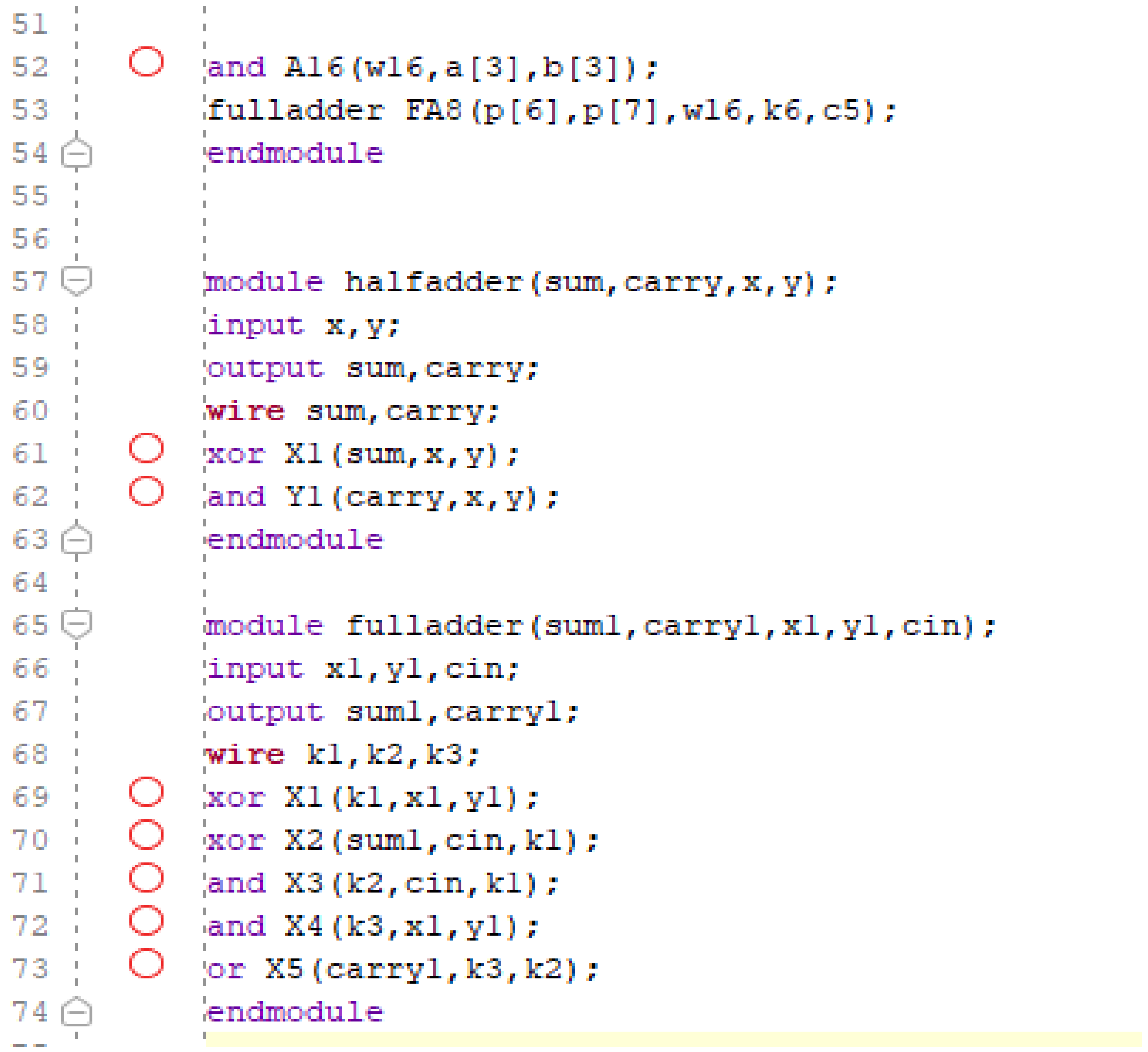
$P_6 = A_3B_3$



Code for 4 Bit

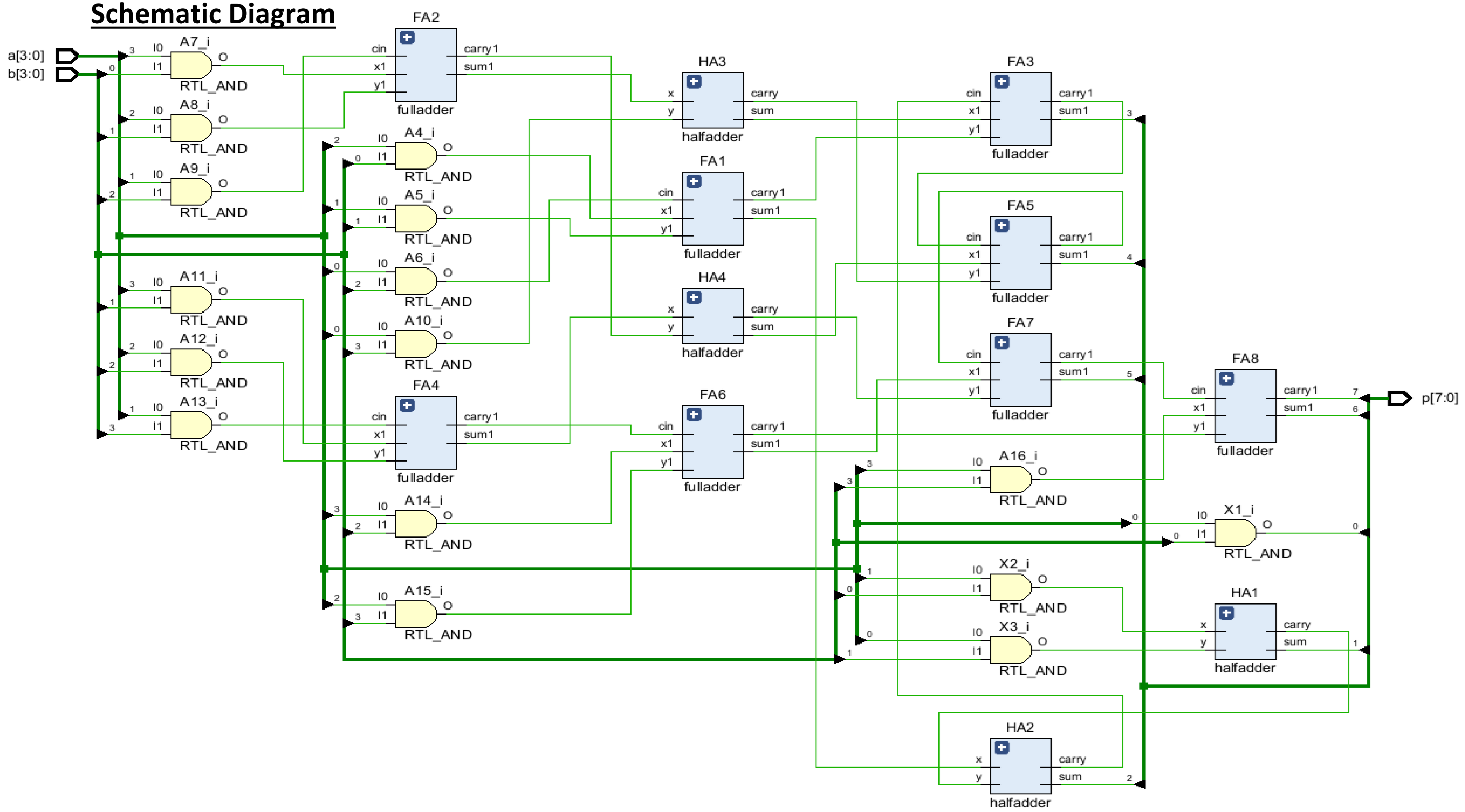


```verilog

module fourbitmultiplier(p,a,b);
input  [3:0]a;
input  [3:0]b;
output  [7:0]p;
wire  [7:0]p;
wire  c1,c2,c3,c4,c5,k1,k2,k3,k4,k5,k6,s1,s2,s3,s4,s5,s6;
wire  w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14,w15,w16;
and X1(p[0],a[0],b[0]);

and X2(w2,a[1],b[0]);
and X3(w3,a[0],b[1]);
halfadder  HA1(p[1],c1,w2,w3);

and A4(w4,a[2],b[0]);
and A5(w5,a[1],b[1]);
and A6(w6,a[0],b[2]);
fulladder  FA1(s1,k1,w4,w5,w6);
halfadder  HA2(p[2],c2,s1,c1);

and A7(w7,a[3],b[0]);
and A8(w8,a[2],b[1]);
and A9(w9,a[1],b[2]);
fulladder  FA2(s2,k2,w7,w8,w9);
and A10(w10,a[0],b[3]);
halfadder  HA3(s3,k3,s2,w10);
fulladder  FA3(p[3],c3,s3,k1,c2);

and A11(w11,a[3],b[1]);
and A12(w12,a[2],b[2]);
and A13(w13,a[1],b[3]);
fulladder  FA4(s4,k4,w11,w12,w13);
halfadder  HA4(s5,k5,s4,k2);
fulladder  FA5(p[4],c4,s5,k3,c3);

and A14(w14,a[3],b[2]);
and A15(w15,a[2],b[3]);
fulladder  FA6(s6,k6,w14,w15,k4);
fulladder  FA7(p[5],c5,s6,k5,c4);

and A16(w16,a[3],b[3]);
fulladder  FA8(p[6],p[7],w16,k6,c5);
endmodule

```

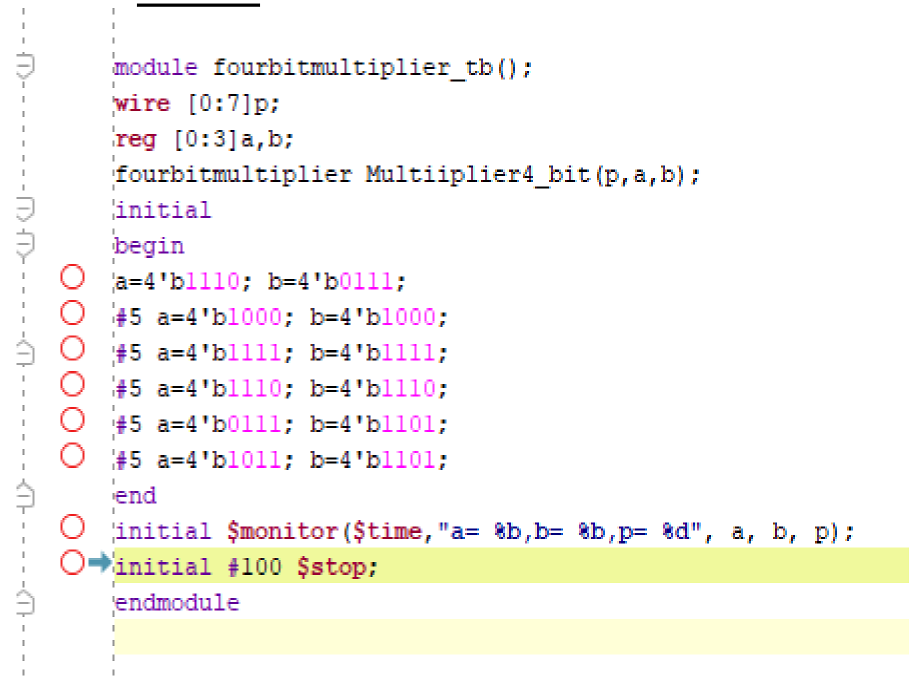


```

and A16(w16,a[3],b[3]);
fulladder FA8(p[6],p[7],w16,k6,c5);
endmodule


module halfadder(sum,carry,x,y);
input x,y;
output sum,carry;
wire sum,carry;
xor X1(sum,x,y);
and Y1(carry,x,y);
endmodule

module fulladder(sum1,carry1,x1,y1,cin);
input x1,y1,cin;
output sum1,carry1;
wire k1,k2,k3;
xor X1(k1,x1,y1);
xor X2(sum1,cin,k1);
and X3(k2,cin,k1);
and X4(k3,x1,y1);
or X5(carry1,k3,k2);
endmodule
```

Schematic Diagram

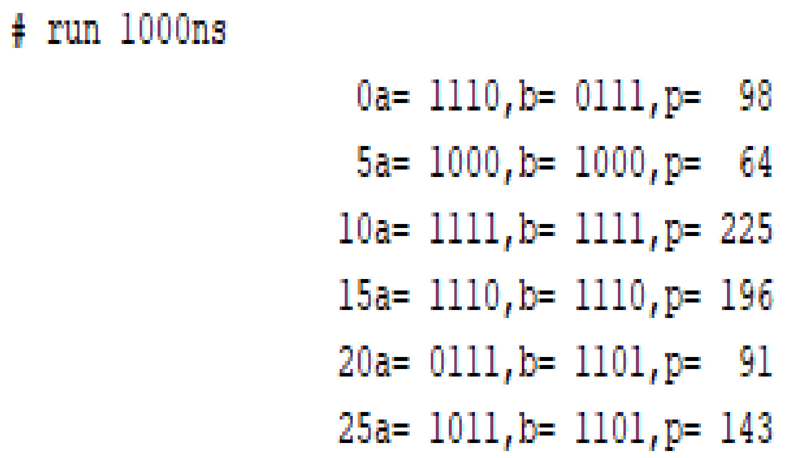


Test Bench



```
module fourbitmultiplier_tb();
wire [0:7]p;
reg [0:3]a,b;
fourbitmultiplier Multiiplier4_bit(p,a,b);
initial
begin
a=4'b1110; b=4'b0111;
#5 a=4'b1000; b=4'b1000;
#5 a=4'b1111; b=4'b1111;
#5 a=4'b1110; b=4'b1110;
#5 a=4'b0111; b=4'b1101;
#5 a=4'b1011; b=4'b1101;
end
initial $monitor($time,"a= %b,b= %b,p= %d", a, b, p);
initial #100 $stop;
endmodule
```

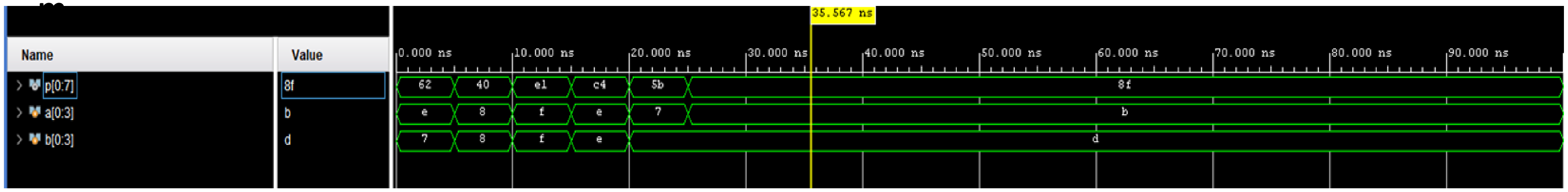
Result

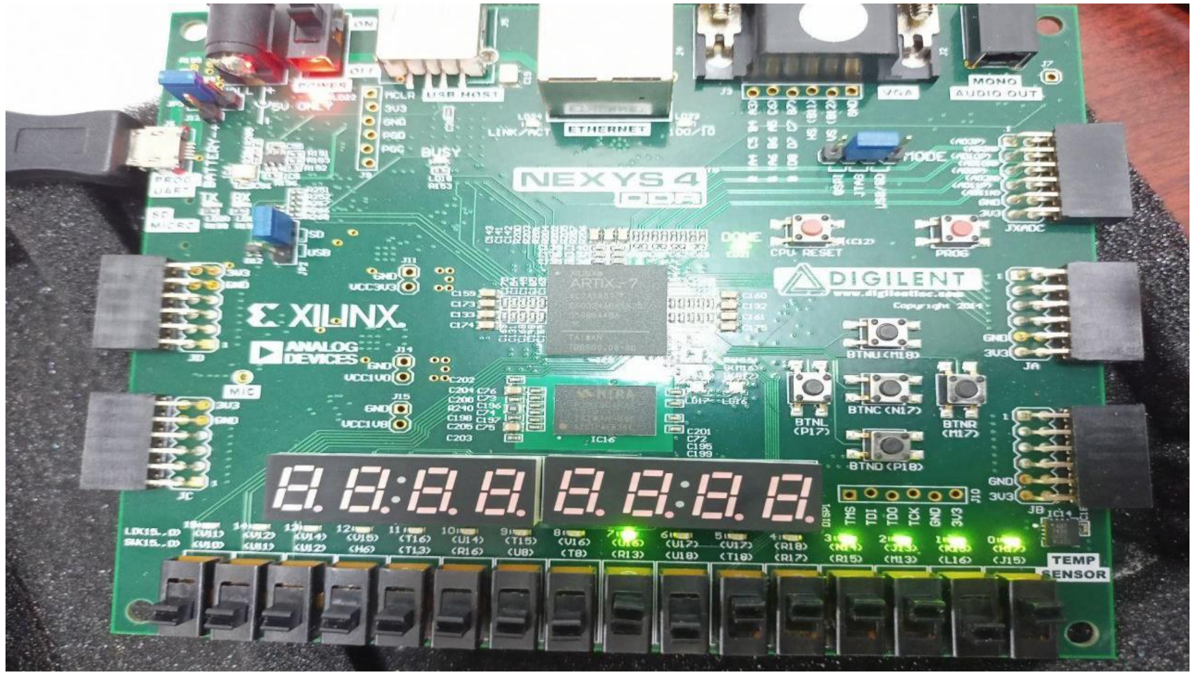


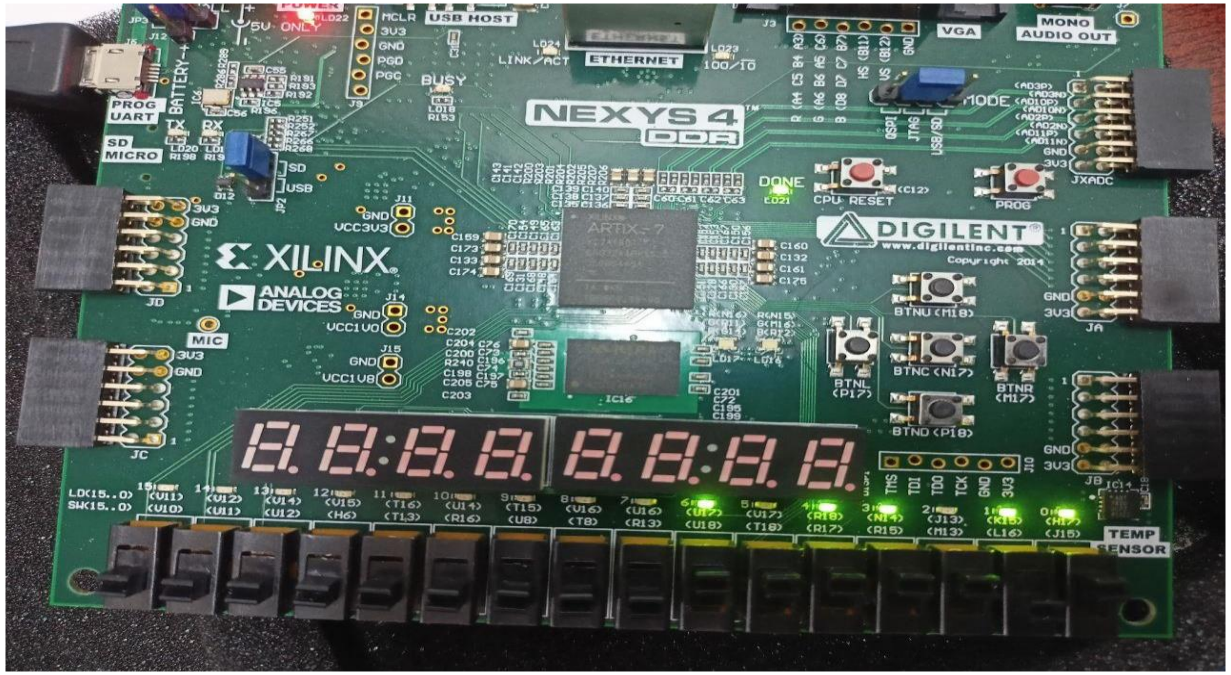
```
# run 1000ns

                 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
```

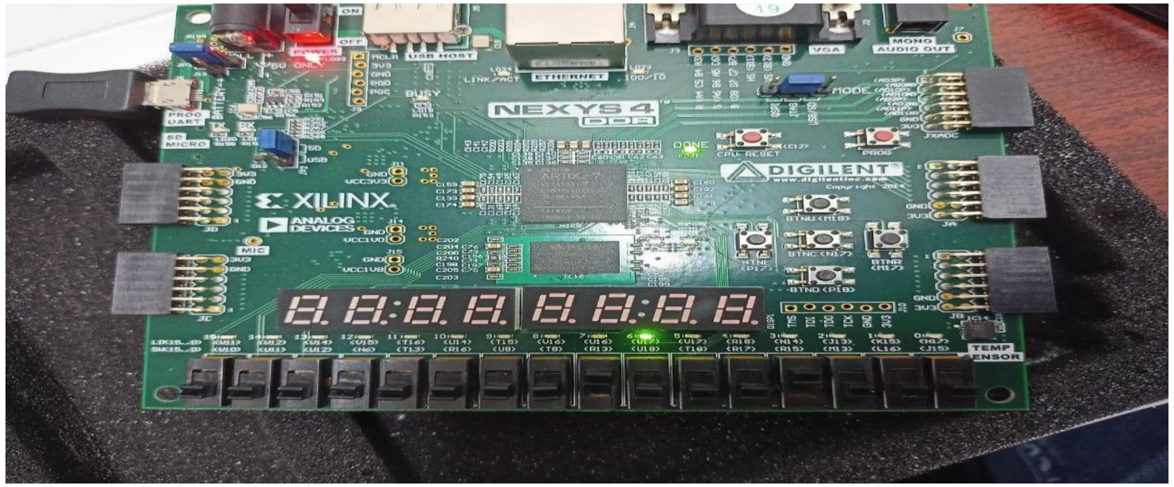
Results are in Decimal number.

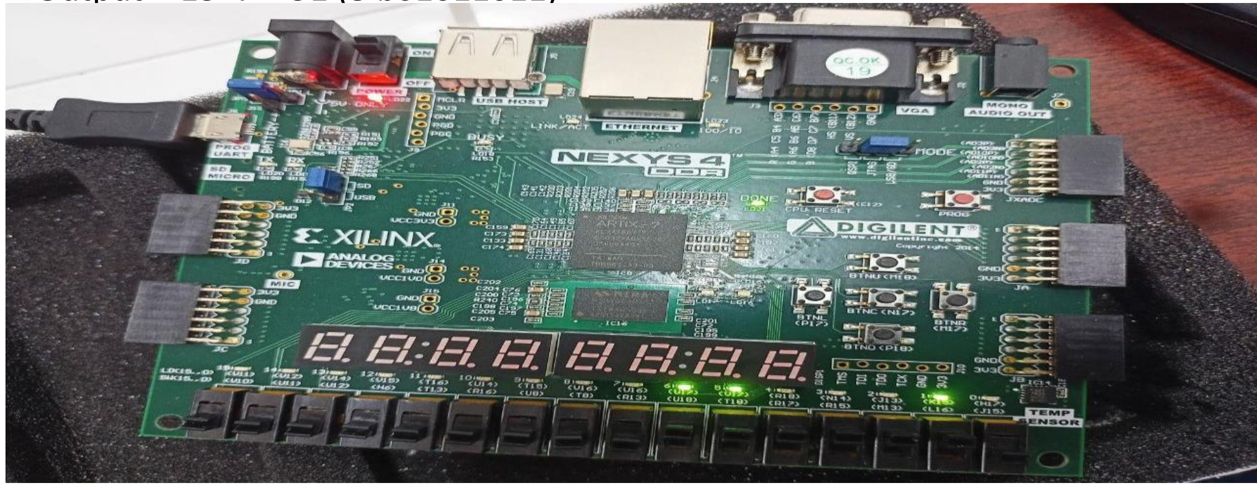
Waveform




A=4’b1101 i.e 13 and B=4’b1011 i.e. 11
Output = 13 * 11 = 143(8’b10001111)


A=4’b1101 i.e. 13 and B=4’b0111 i.e. 7
Output = 13*7 = 91 (8’b01011011)

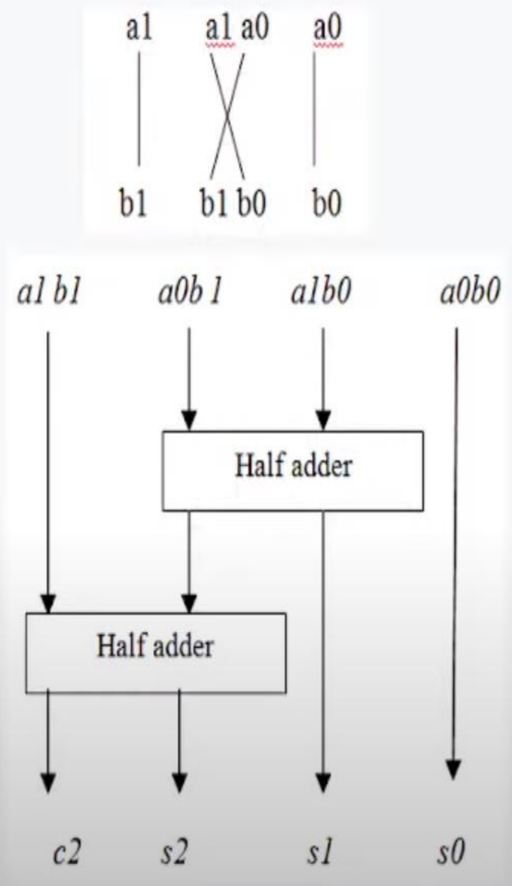

A=4’b1000 i.e. 8 and B=4’b1000 i.e. 8
Output = 8*8 = 64(8’b01000000)


A=4’b1110 i.e. 14 and B=4’b0111 i.e. 7
Output = 14 * 7 = 98(8’b01100010)

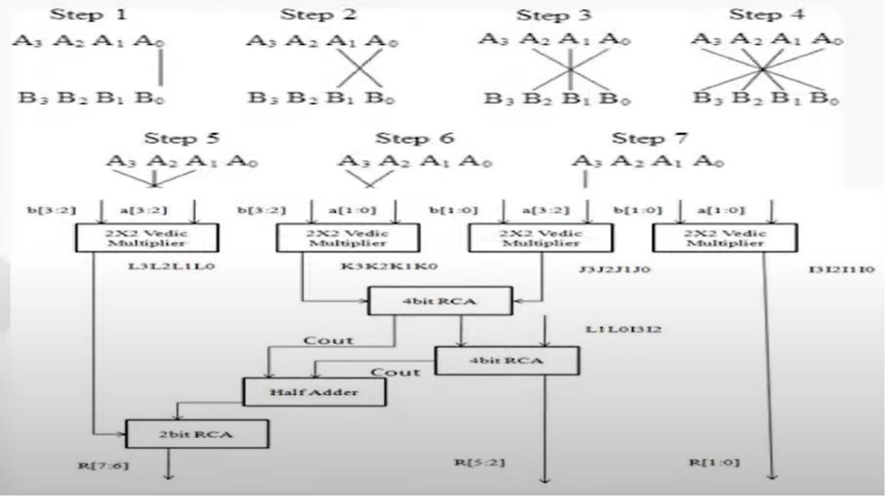
8 Bit Vedic Multiplier

								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

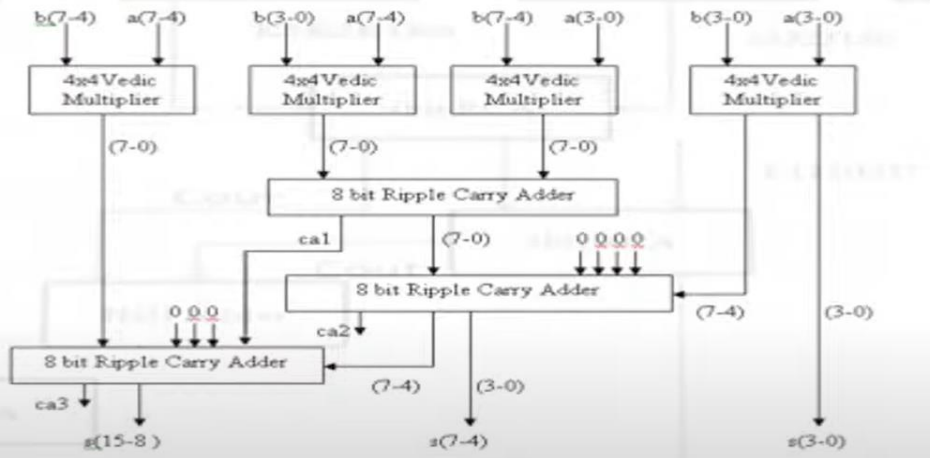
2-bit Vedic Multiplier



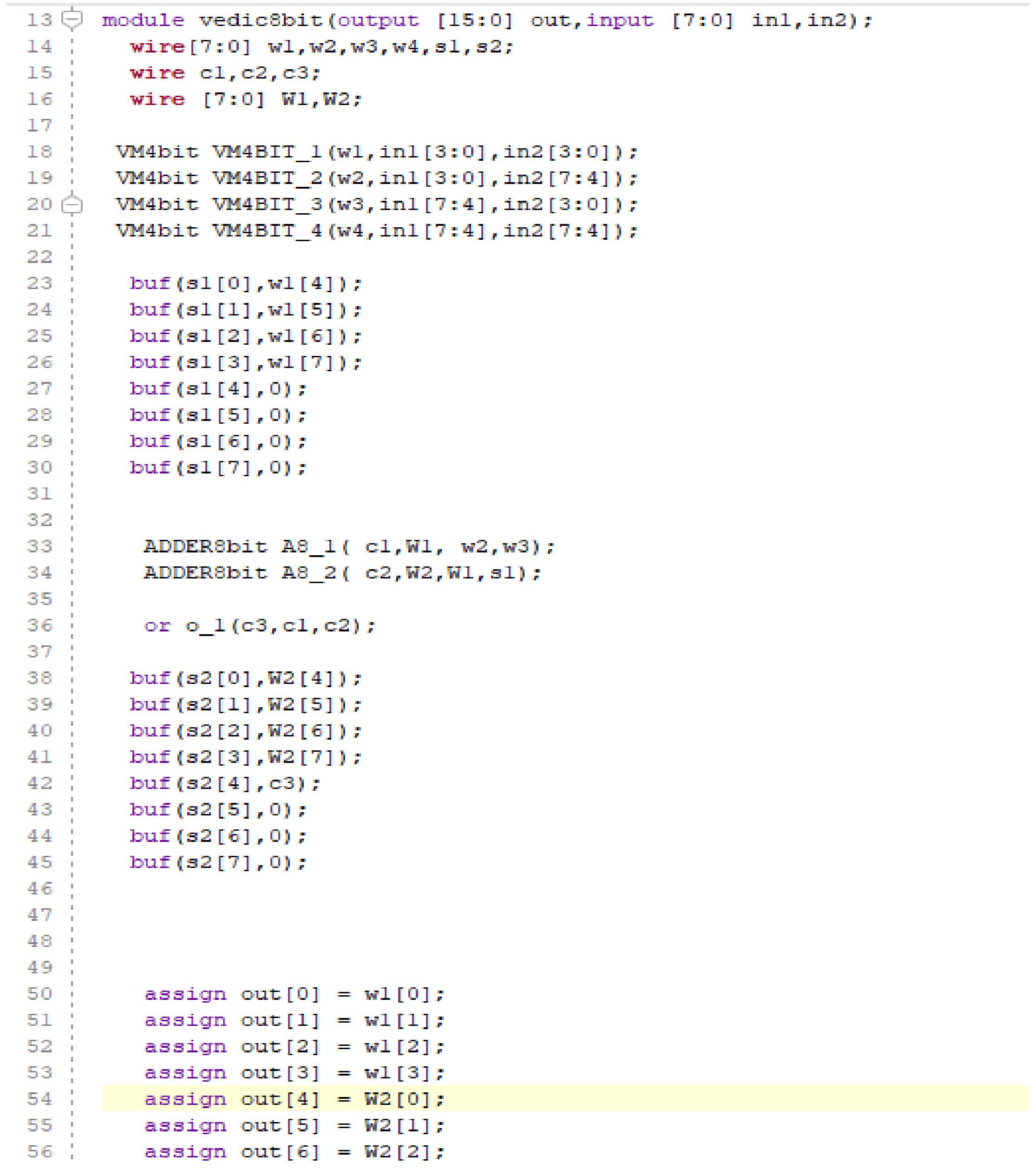
4-bit Vedic Multiplier



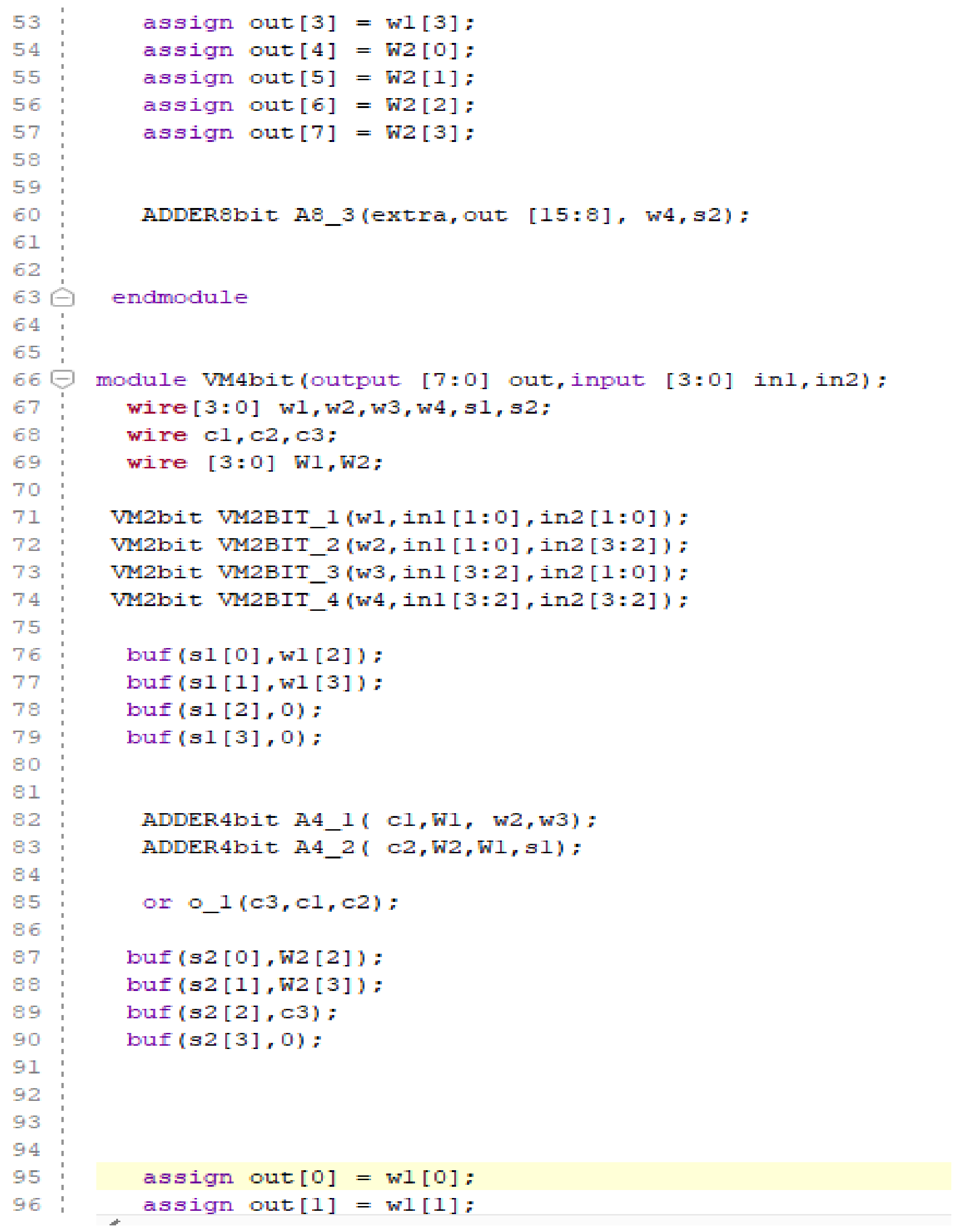
8-bit Vedic Multiplier



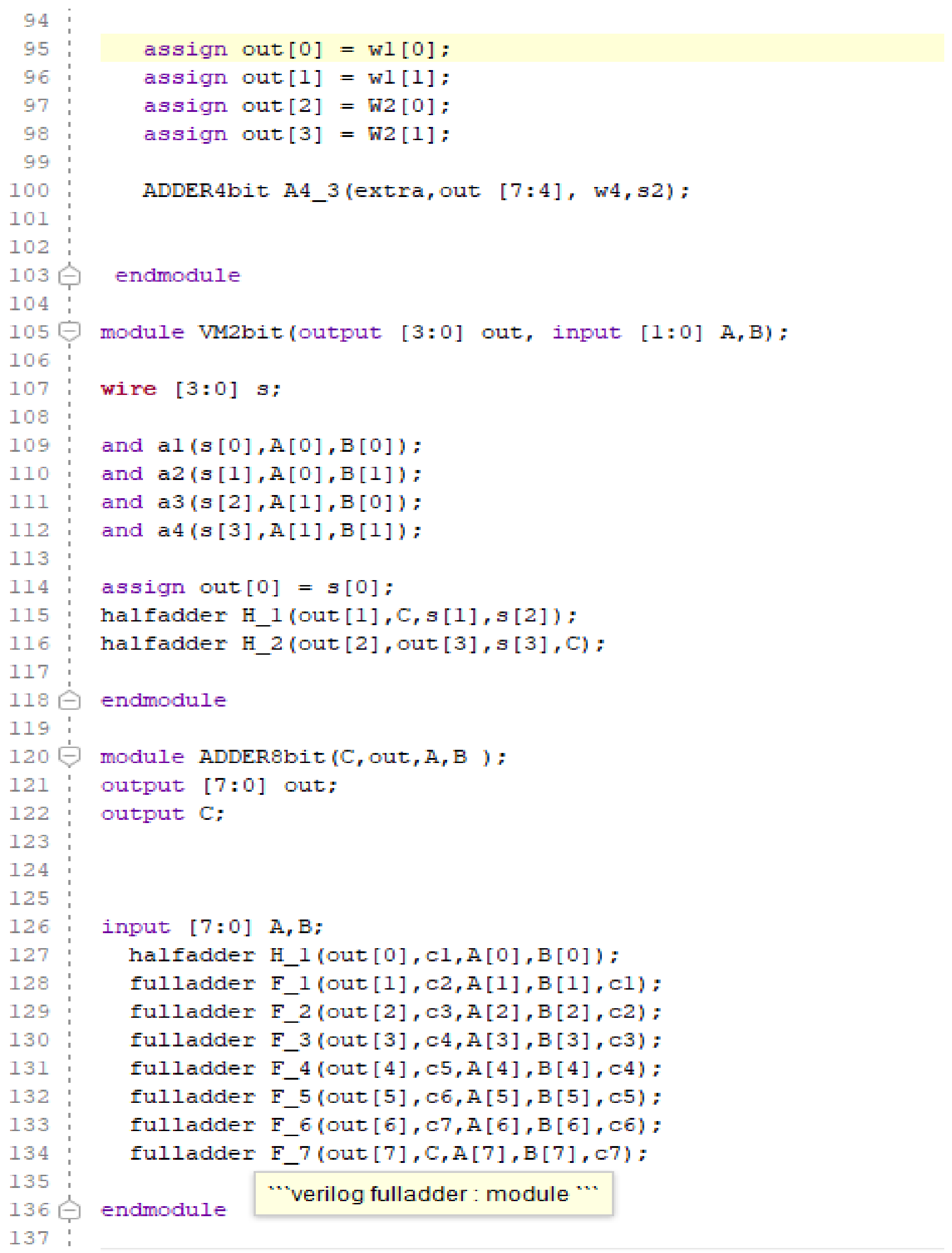
Code for 8-bit



```
module vedic8bit(output [15:0] out,input [7:0] in1,in2);
  wire[7:0] w1,w2,w3,w4,s1,s2;
  wire c1,c2,c3;
  wire [7:0] W1,W2;

 VM4bit VM4BIT_1(w1,in1[3:0],in2[3:0]);
 VM4bit VM4BIT_2(w2,in1[3:0],in2[7:4]);
 VM4bit VM4BIT_3(w3,in1[7:4],in2[3:0]);
 VM4bit VM4BIT_4(w4,in1[7:4],in2[7:4]);

  buf(s1[0],w1[4]);
  buf(s1[1],w1[5]);
  buf(s1[2],w1[6]);
  buf(s1[3],w1[7]);
  buf(s1[4],0);
  buf(s1[5],0);
  buf(s1[6],0);
  buf(s1[7],0);


   ADDER8bit A8_1( c1,W1, w2,w3);
   ADDER8bit A8_2( c2,W2,W1,s1);

   or o_1(c3,c1,c2);

  buf(s2[0],W2[4]);
  buf(s2[1],W2[5]);
  buf(s2[2],W2[6]);
  buf(s2[3],W2[7]);
  buf(s2[4],c3);
  buf(s2[5],0);
  buf(s2[6],0);
  buf(s2[7],0);




   assign out[0] = w1[0];
   assign out[1] = w1[1];
   assign out[2] = w1[2];
   assign out[3] = w1[3];
   assign out[4] = W2[0];
   assign out[5] = W2[1];
   assign out[6] = W2[2];
```

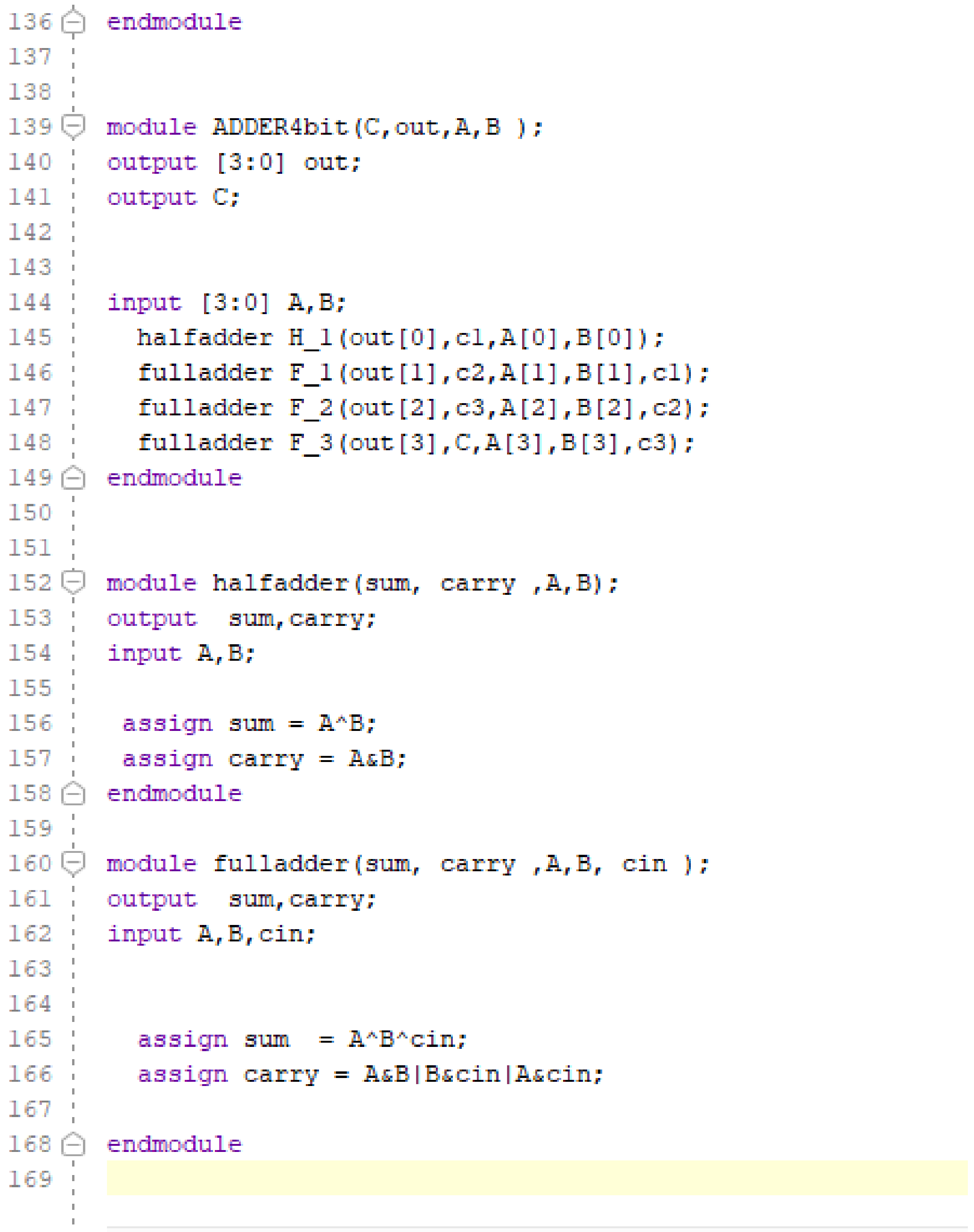


```
   assign out[3] = w1[3];
   assign out[4] = W2[0];
   assign out[5] = W2[1];
   assign out[6] = W2[2];
   assign out[7] = W2[3];


   ADDER8bit A8_3(extra,out [15:8], w4,s2);


 endmodule


module VM4bit(output [7:0] out,input [3:0] in1,in2);
  wire[3:0] w1,w2,w3,w4,s1,s2;
  wire c1,c2,c3;
  wire [3:0] W1,W2;

 VM2bit VM2BIT_1(w1,in1[1:0],in2[1:0]);
 VM2bit VM2BIT_2(w2,in1[1:0],in2[3:2]);
 VM2bit VM2BIT_3(w3,in1[3:2],in2[1:0]);
 VM2bit VM2BIT_4(w4,in1[3:2],in2[3:2]);

  buf(s1[0],w1[2]);
  buf(s1[1],w1[3]);
  buf(s1[2],0);
  buf(s1[3],0);


   ADDER4bit A4_1( c1,W1, w2,w3);
   ADDER4bit A4_2( c2,W2,W1,s1);

   or o_1(c3,c1,c2);

  buf(s2[0],W2[2]);
  buf(s2[1],W2[3]);
  buf(s2[2],c3);
  buf(s2[3],0);




   assign out[0] = w1[0];
   assign out[1] = w1[1];
```



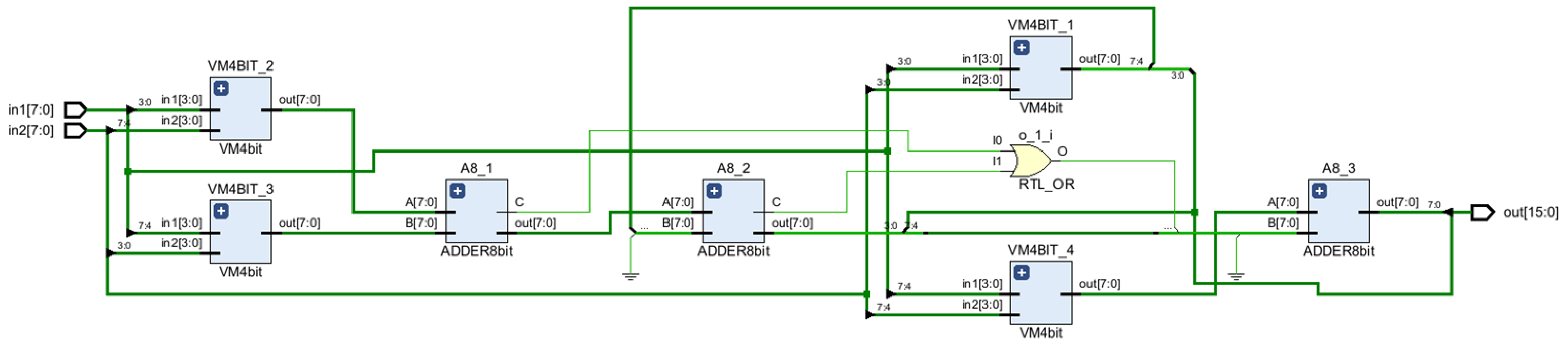
```verilog
   assign out[0] = w1[0];
   assign out[1] = w1[1];
   assign out[2] = W2[0];
   assign out[3] = W2[1];

   ADDER4bit A4_3(extra,out [7:4], w4,s2);


 endmodule

module VM2bit(output [3:0] out, input [1:0] A,B);

wire [3:0] s;

and a1(s[0],A[0],B[0]);
and a2(s[1],A[0],B[1]);
and a3(s[2],A[1],B[0]);
and a4(s[3],A[1],B[1]);

assign out[0] = s[0];
halfadder H_1(out[1],C,s[1],s[2]);
halfadder H_2(out[2],out[3],s[3],C);

endmodule

module ADDER8bit(C,out,A,B );
output [7:0] out;
output C;



input [7:0] A,B;
  halfadder H_1(out[0],c1,A[0],B[0]);
  fulladder F_1(out[1],c2,A[1],B[1],c1);
  fulladder F_2(out[2],c3,A[2],B[2],c2);
  fulladder F_3(out[3],c4,A[3],B[3],c3);
  fulladder F_4(out[4],c5,A[4],B[4],c4);
  fulladder F_5(out[5],c6,A[5],B[5],c5);
  fulladder F_6(out[6],c7,A[6],B[6],c6);
  fulladder F_7(out[7],C,A[7],B[7],c7);

endmodule
```

```verilog fulladder : module ```

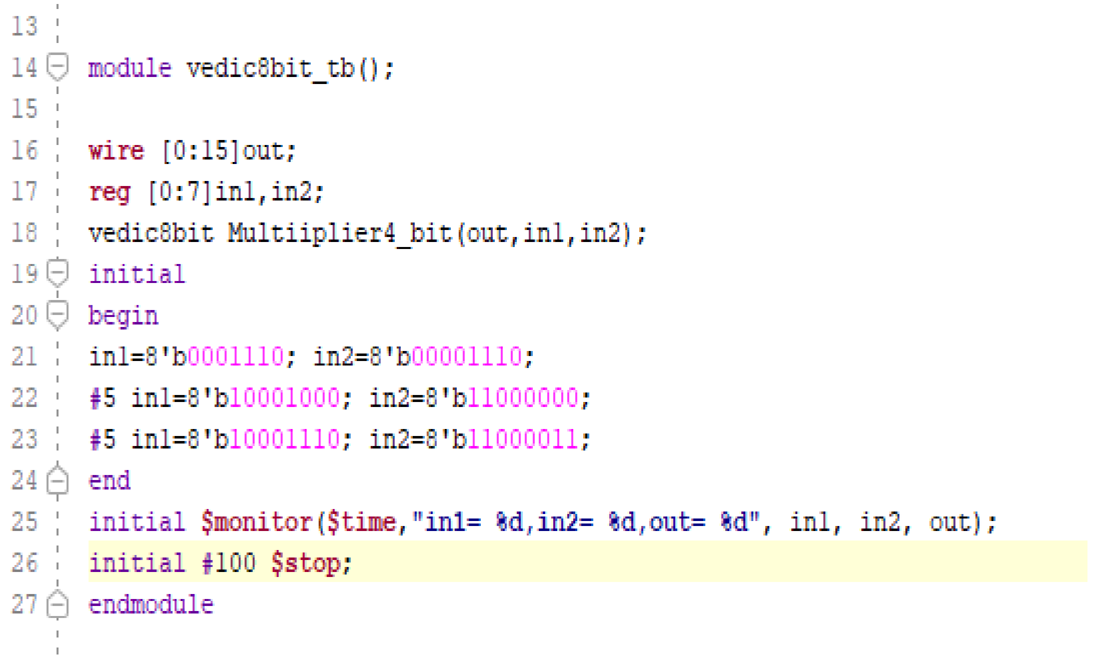


```verilog
endmodule


module ADDER4bit(C,out,A,B );
output [3:0] out;
output C;


input [3:0] A,B;
  halfadder H_1(out[0],c1,A[0],B[0]);
  fulladder F_1(out[1],c2,A[1],B[1],c1);
  fulladder F_2(out[2],c3,A[2],B[2],c2);
  fulladder F_3(out[3],C,A[3],B[3],c3);
endmodule


module halfadder(sum, carry ,A,B);
output  sum,carry;
input A,B;

 assign sum = A^B;
 assign carry = A&B;
endmodule

module fulladder(sum, carry ,A,B, cin );
output  sum,carry;
input A,B,cin;


  assign sum  = A^B^cin;
  assign carry = A&B|B&cin|A&cin;

endmodule
```

Schematic Diagram

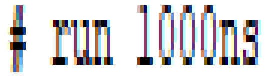


Test Bench

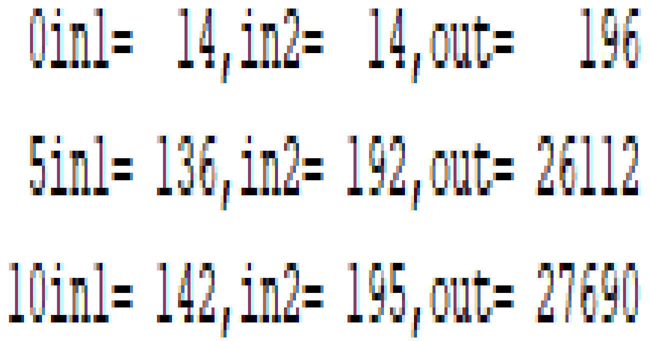


```
module vedic8bit_tb();

wire [0:15]out;
reg [0:7]in1,in2;
vedic8bit Multiiplier4_bit(out,in1,in2);
initial
begin
in1=8'b0001110; in2=8'b00001110;
#5 in1=8'b10001000; in2=8'b11000000;
#5 in1=8'b10001110; in2=8'b11000011;
end
initial $monitor($time,"in1= %d,in2= %d,out= %d", in1, in2, out);
initial #100 $stop;
endmodule
```

Result



```
# run 1000ns
```



```
 0in1=  14,in2=  14,out=   196
 5in1= 136,in2= 192,out= 26112
10in1= 142,in2= 195,out= 27690
```