DIGITAL DESIGN LAB 4

DIVYAM PATEL

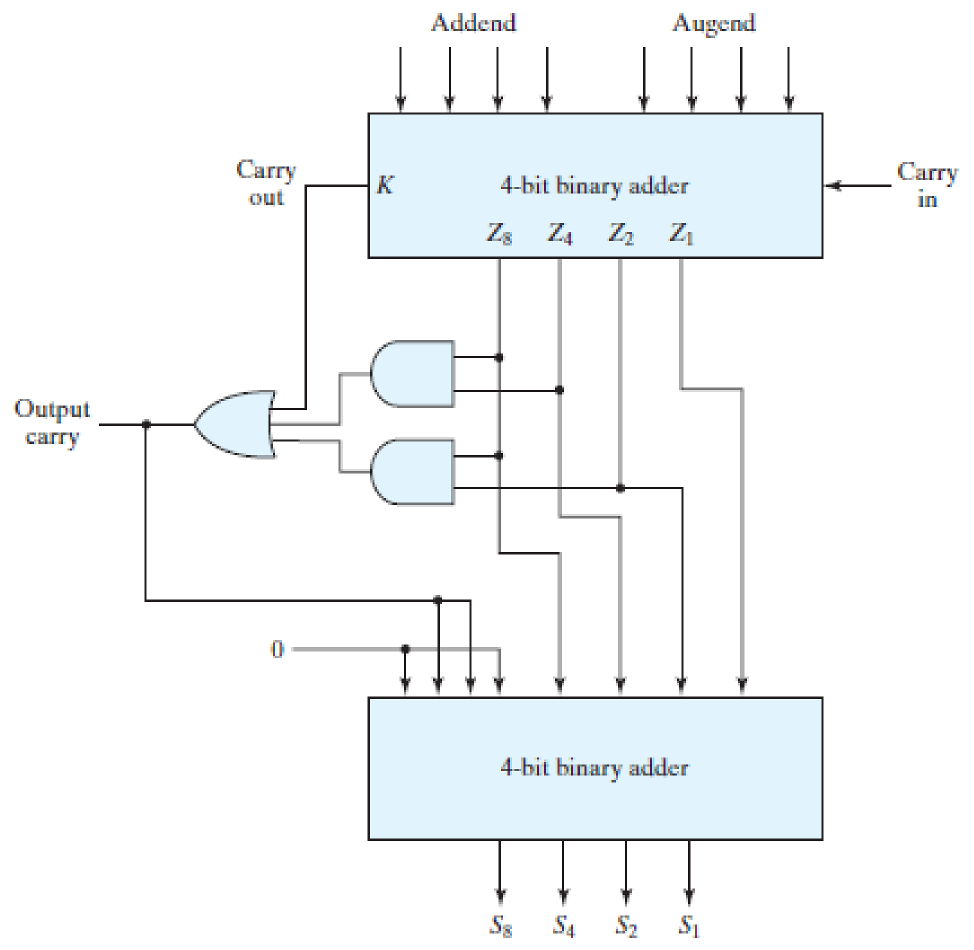
B20EE082

1. Write a Program to implement a
a) BCD Adder Unit
b) BCD Subtractor Unit

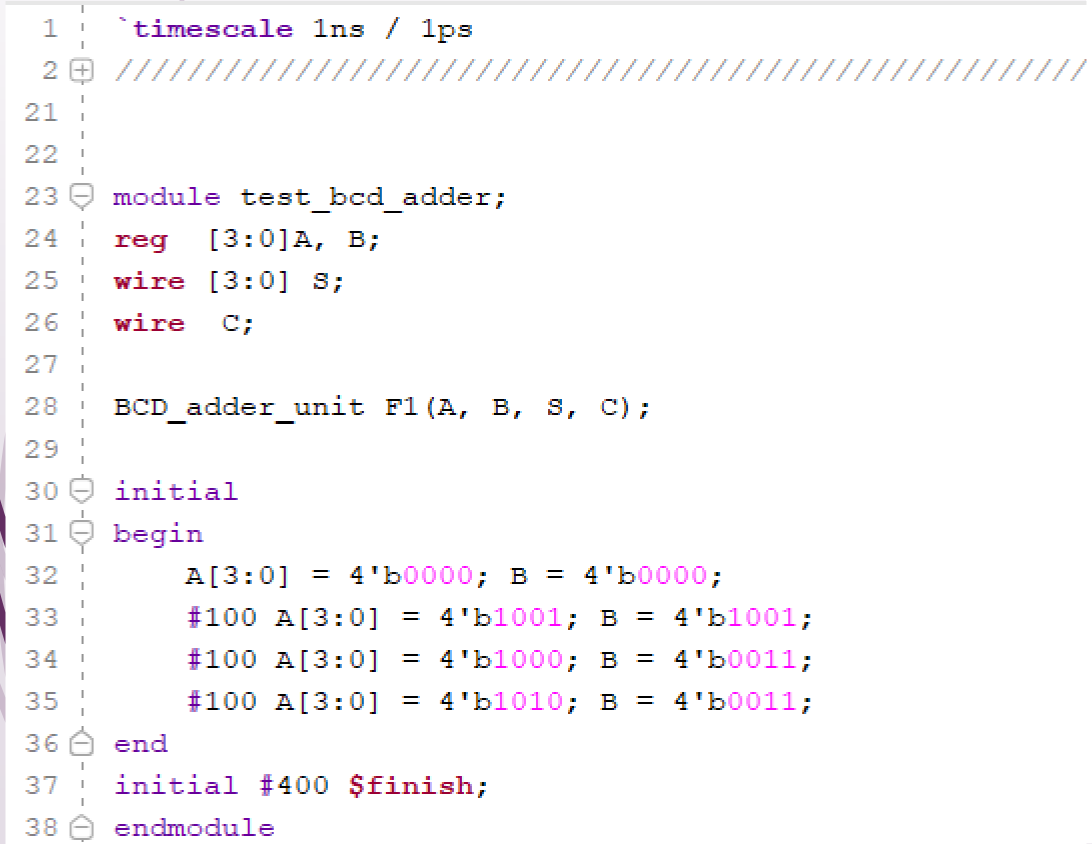
BCD ADDER

Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

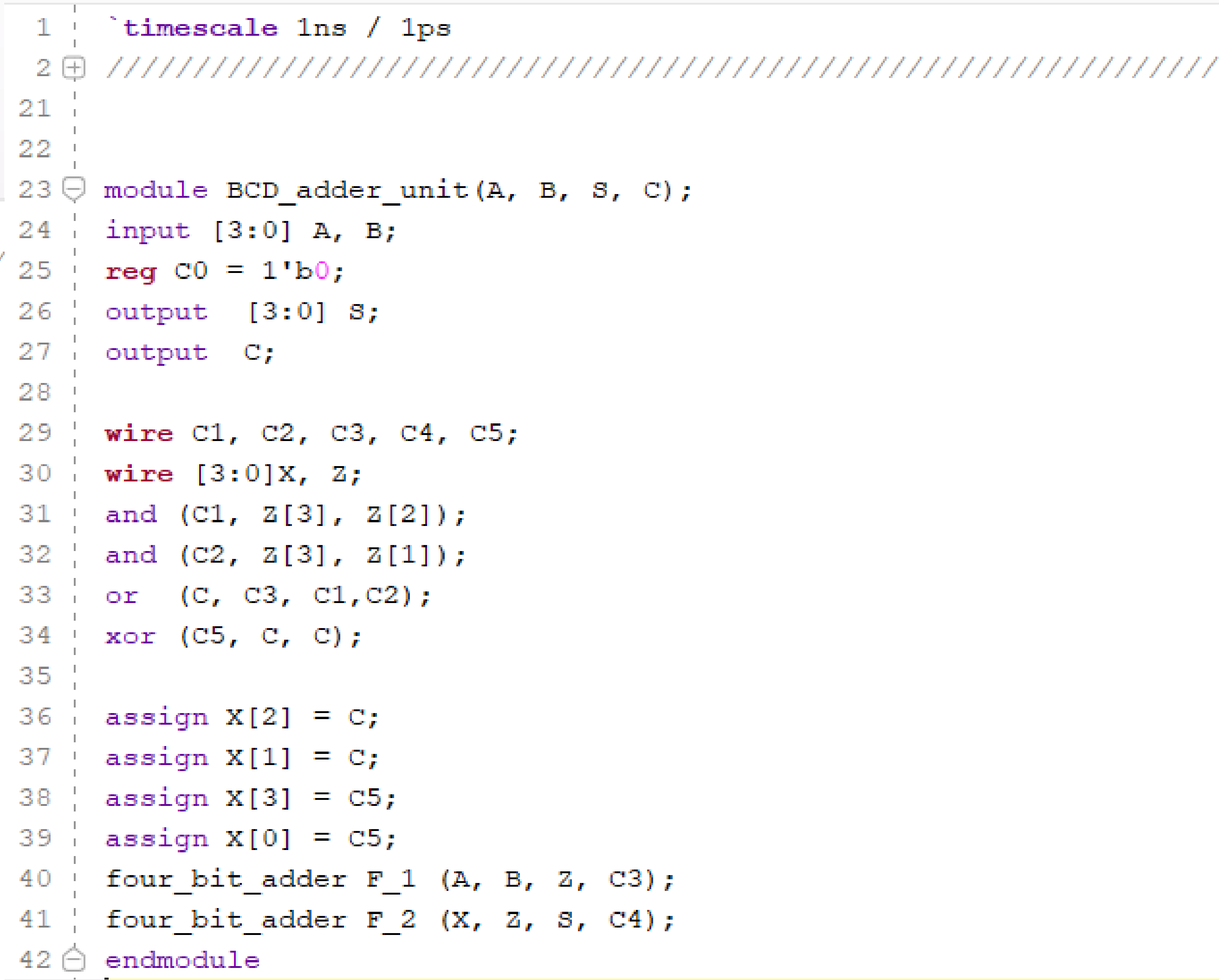


CODE

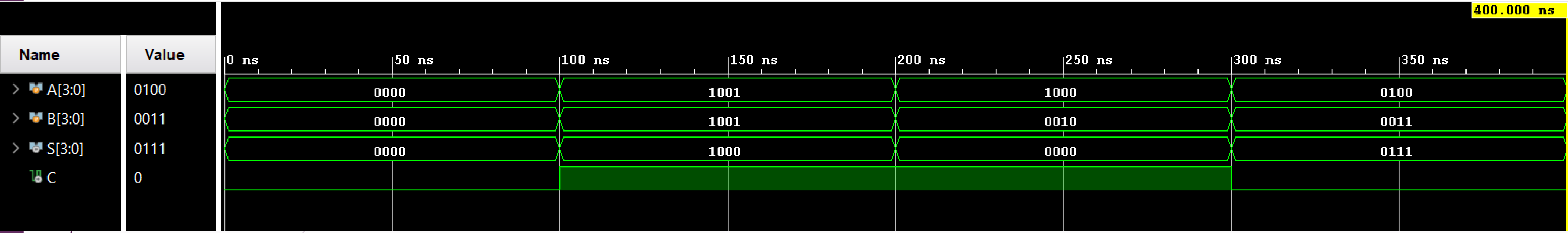


```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////


module test_bcd_adder;
reg  [3:0]A, B;
wire [3:0] S;
wire  C;

BCD_adder_unit F1(A, B, S, C);

initial
begin
    A[3:0] = 4'b0000; B = 4'b0000;
    #100 A[3:0] = 4'b1001; B = 4'b1001;
    #100 A[3:0] = 4'b1000; B = 4'b0011;
    #100 A[3:0] = 4'b1010; B = 4'b0011;
end
initial #400 $finish;
endmodule
```

TESTBENCH

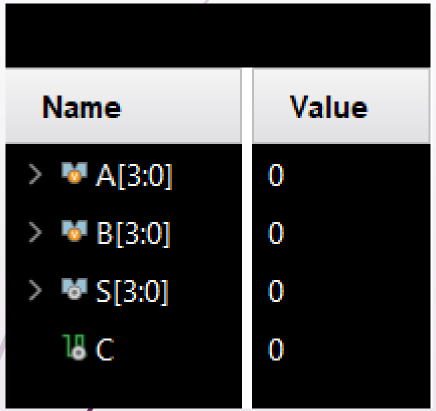


```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////////////////


module BCD_adder_unit(A, B, S, C);
input [3:0] A, B;
reg C0 = 1'b0;
output  [3:0] S;
output  C;

wire C1, C2, C3, C4, C5;
wire [3:0]X, Z;
and (C1, Z[3], Z[2]);
and (C2, Z[3], Z[1]);
or  (C, C3, C1,C2);
xor (C5, C, C);

assign X[2] = C;
assign X[1] = C;
assign X[3] = C5;
assign X[0] = C5;
four_bit_adder F_1 (A, B, Z, C3);
four_bit_adder F_2 (X, Z, S, C4);
endmodule
```



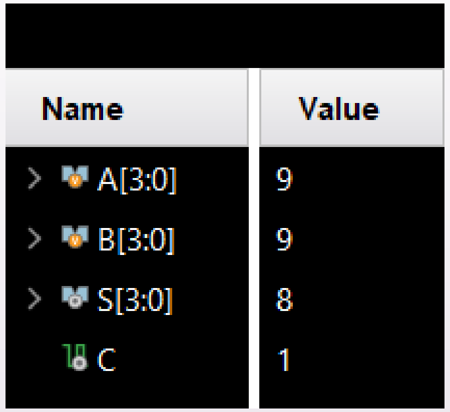
400.000 ns

Name	Value	0 ns	100 ns	200 ns	300 ns
A[3:0]	0100	0000	1001	1000	0100
B[3:0]	0011	0000	1001	0010	0011
S[3:0]	0111	0000	1000	0000	0111
C	0				



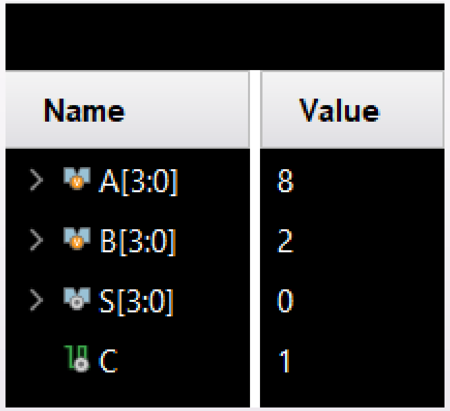
Name	Value
A[3:0]	0
B[3:0]	0
S[3:0]	0
C	0

A: 0000, B: 0000, Sum S: 0000 and Carry C: 0 for the BCD adder (i.e. 0 + 0 = 0 0)



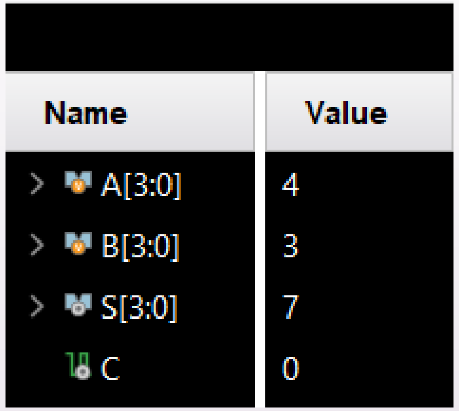
Name	Value
A[3:0]	9
B[3:0]	9
S[3:0]	8
C	1

A: 1001, B: 1001, Sum S: 1000 and Carry C: 1 for the BCD adder (i.e. 9 + 9 = 1 8)



Name	Value
A[3:0]	8
B[3:0]	2
S[3:0]	0
C	1

A: 1000, B: 0010, Sum S: 0000 and Carry C: 1 for the BCD adder (i.e. 8 + 2 = 1 0)



Name	Value
A[3:0]	4
B[3:0]	3
S[3:0]	7
C	0

A: 0100, B: 0011, Sum S: 0111 and Carry C: 1 for the BCD adder (i.e. 8 + 2 = 1 0)

BCD SUBTRACTOR

- Find the 10’s complement of a negative number
- Add two numbers using BCD addition
- If carry is not generated find the 10s Complement of the result.

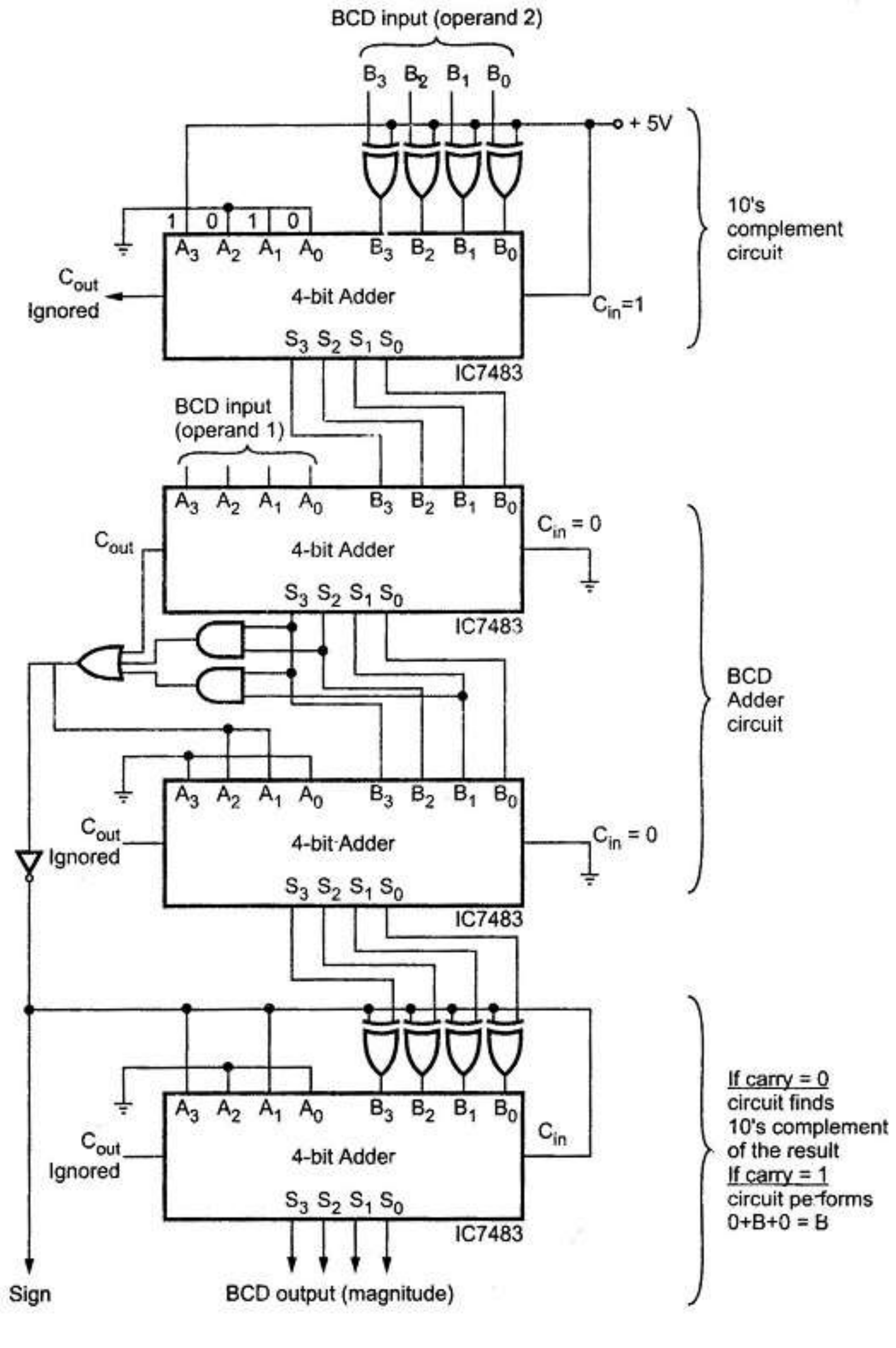
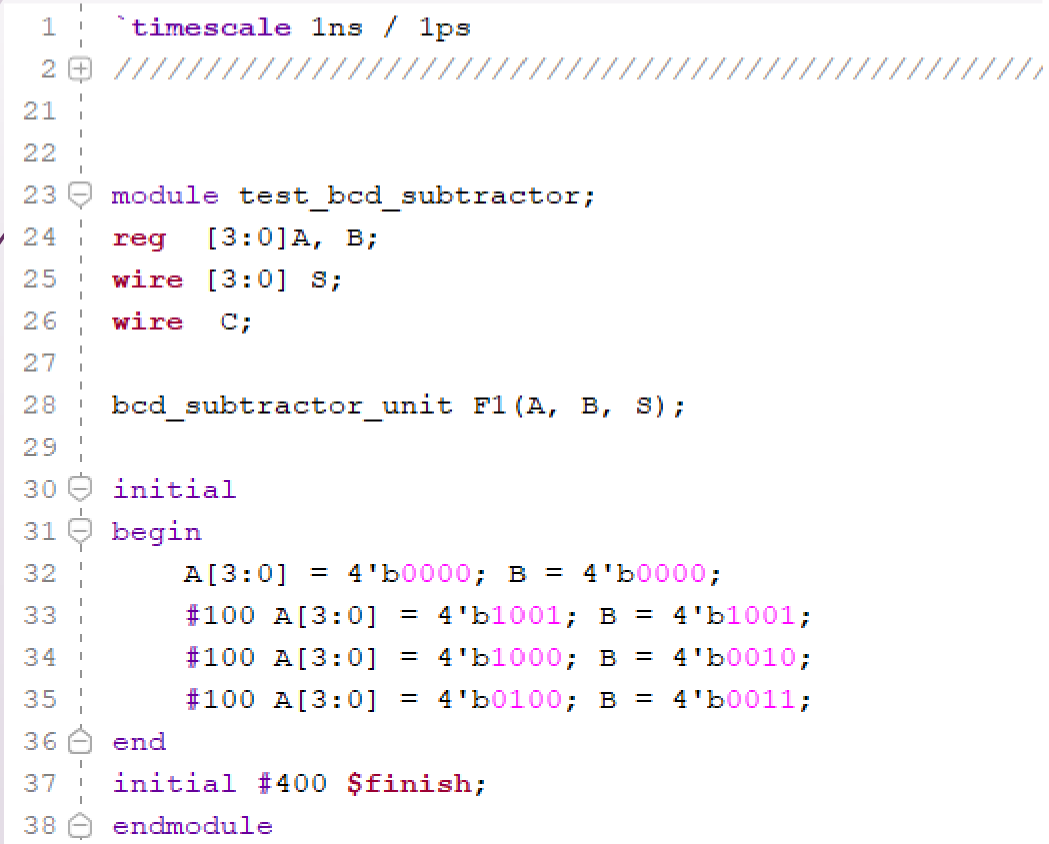


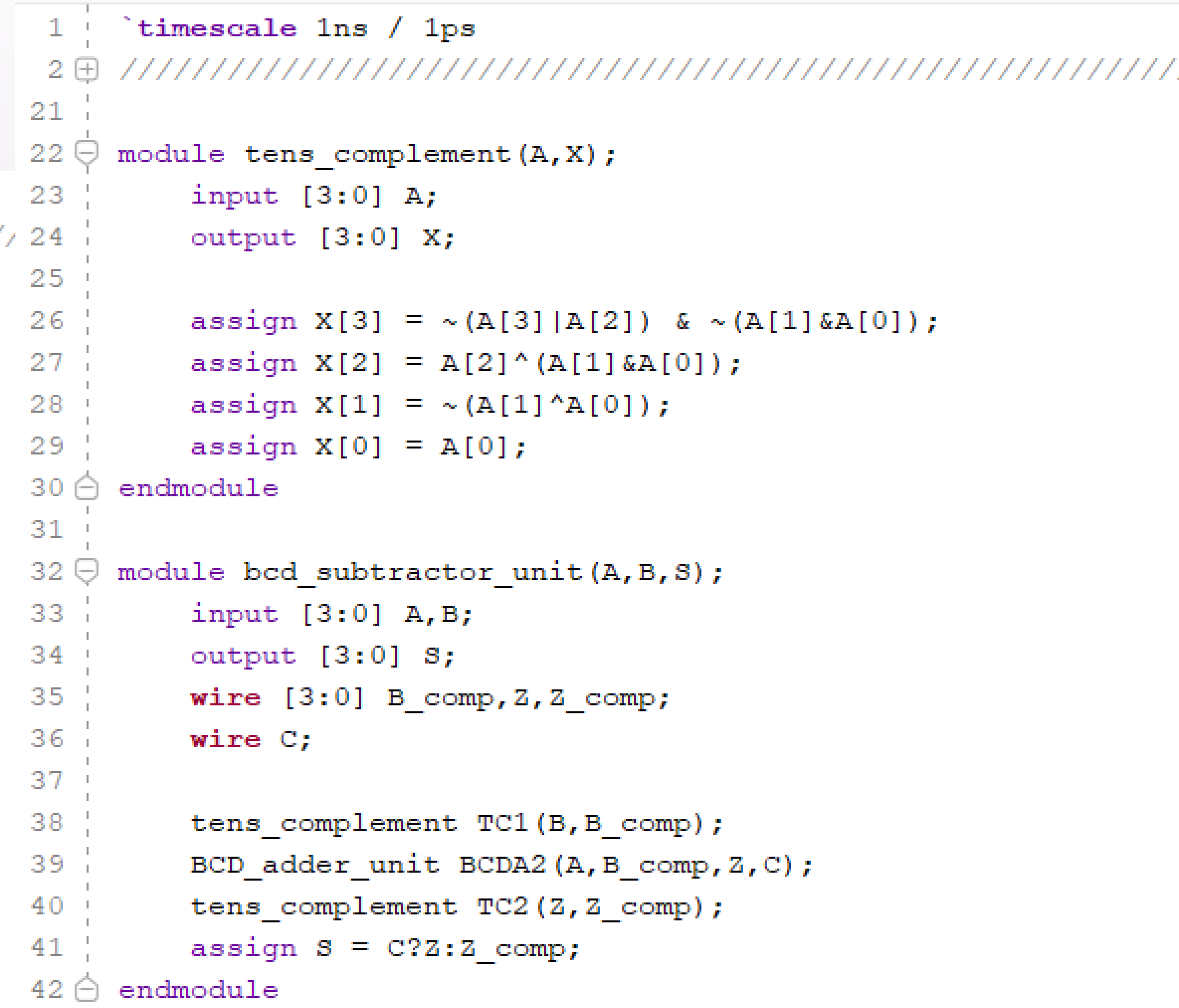
Fig. 3.35 4-bit BCD subtractor using 10's complement method

TESTBENCH

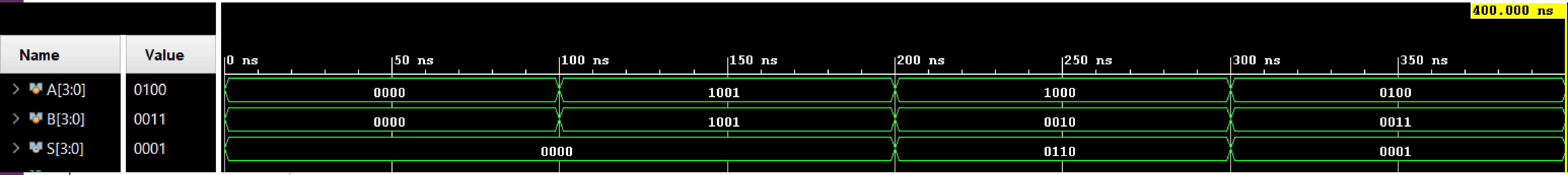


```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////


module test_bcd_subtractor;
reg  [3:0]A, B;
wire [3:0] S;
wire  C;

bcd_subtractor_unit F1(A, B, S);

initial
begin
    A[3:0] = 4'b0000; B = 4'b0000;
    #100 A[3:0] = 4'b1001; B = 4'b1001;
    #100 A[3:0] = 4'b1000; B = 4'b0010;
    #100 A[3:0] = 4'b0100; B = 4'b0011;
end
initial #400 $finish;
endmodule
```

CODE

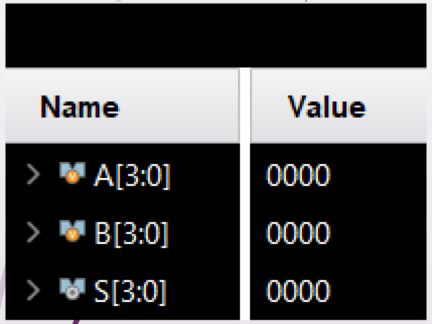


```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////

module tens_complement(A,X);
    input [3:0] A;
    output [3:0] X;

    assign X[3] = ~(A[3]|A[2]) & ~(A[1]&A[0]);
    assign X[2] = A[2]^(A[1]&A[0]);
    assign X[1] = ~(A[1]^A[0]);
    assign X[0] = A[0];
endmodule

module bcd_subtractor_unit(A,B,S);
    input [3:0] A,B;
    output [3:0] S;
    wire [3:0] B_comp,Z,Z_comp;
    wire C;

    tens_complement TC1(B,B_comp);
    BCD_adder_unit BCDA2(A,B_comp,Z,C);
    tens_complement TC2(Z,Z_comp);
    assign S = C?Z:Z_comp;
endmodule
```

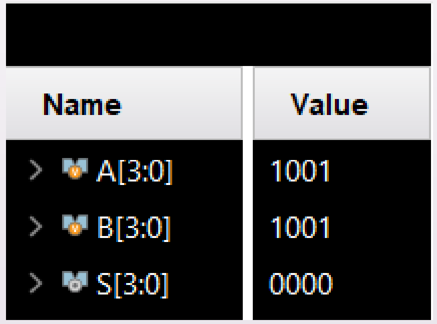


Name	Value
A[3:0]	0100
B[3:0]	0011
S[3:0]	0001

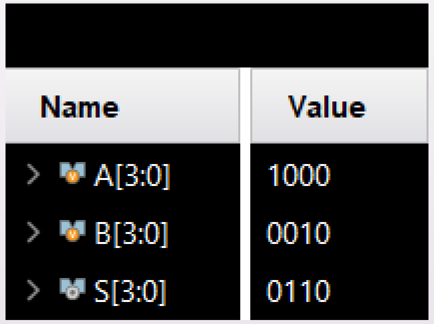
```
400.000 ns
0 ns      50 ns      100 ns      150 ns      200 ns      250 ns      300 ns      350 ns
A:  0000                 1001                   1000                   0100
B:  0000                 1001                   0010                   0011
S:             0000                             0110                   0001
```



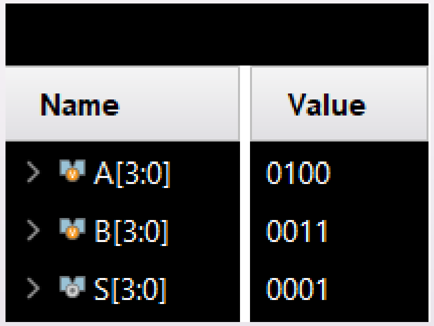
Name	Value
A[3:0]	0000
B[3:0]	0000
S[3:0]	0000



Name	Value
A[3:0]	1001
B[3:0]	1001
S[3:0]	0000

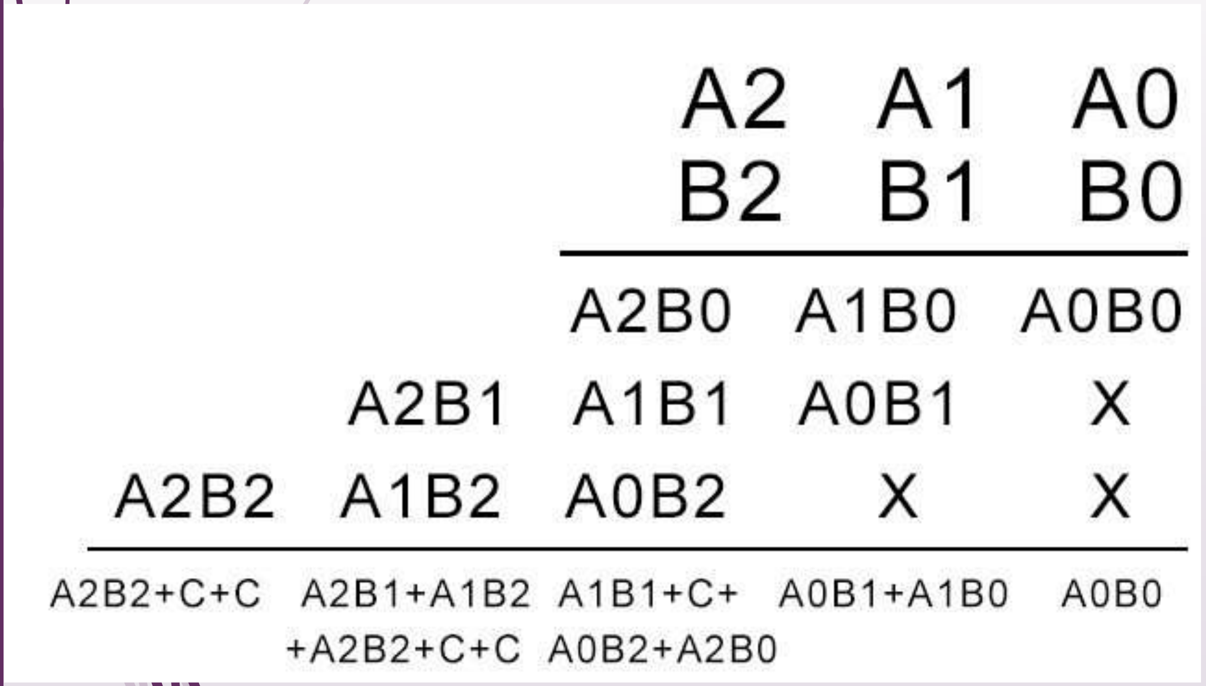


Name	Value
A[3:0]	1000
B[3:0]	0010
S[3:0]	0110



Name	Value
A[3:0]	0100
B[3:0]	0011
S[3:0]	0001

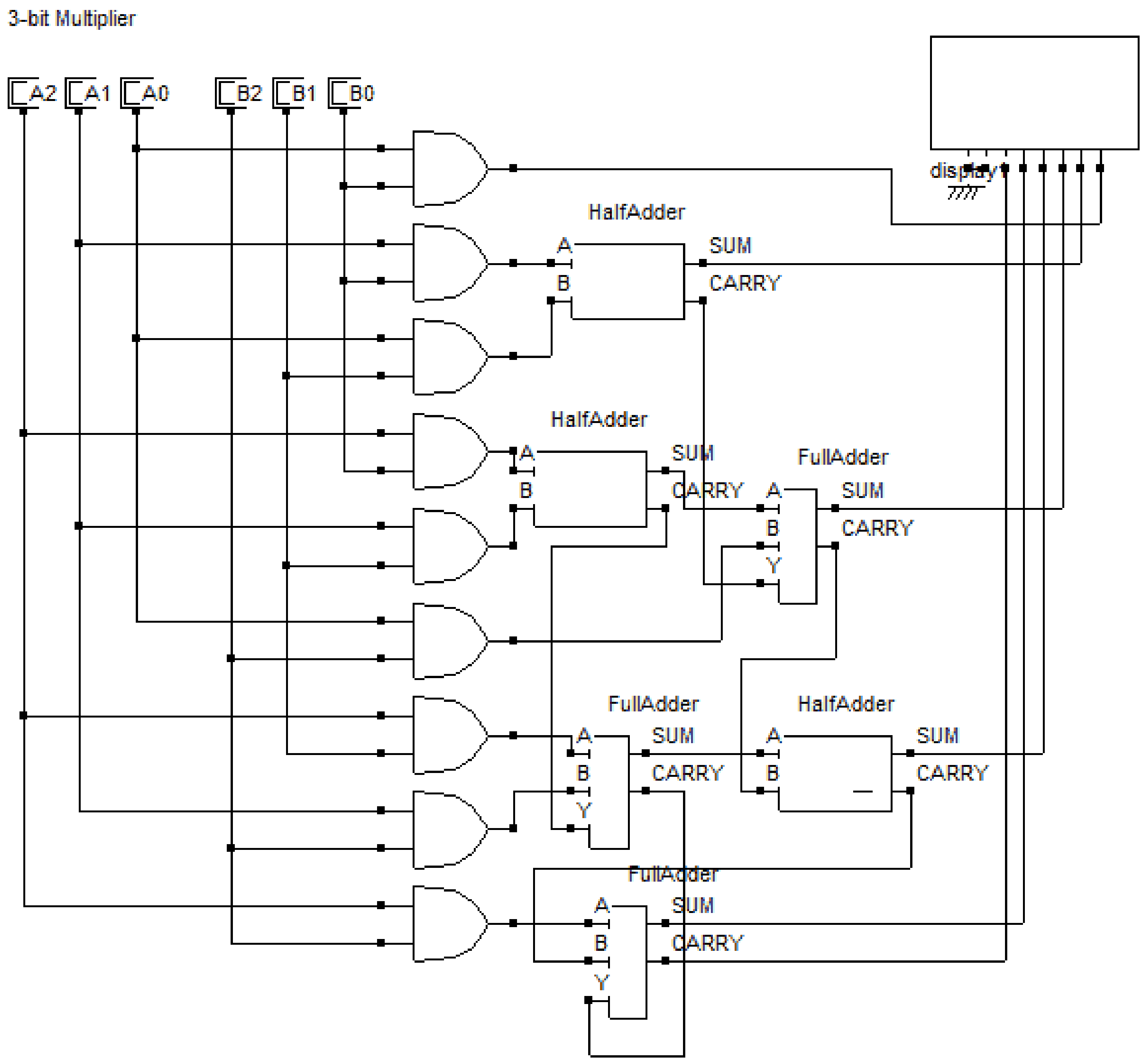
2. Write a Program to implement a
a) Binary Multiplier (3 bit X 3 bit): Maximum product = 7X7 = 49
use input A = A0, A1, A2
use input B = B0,B1, B2



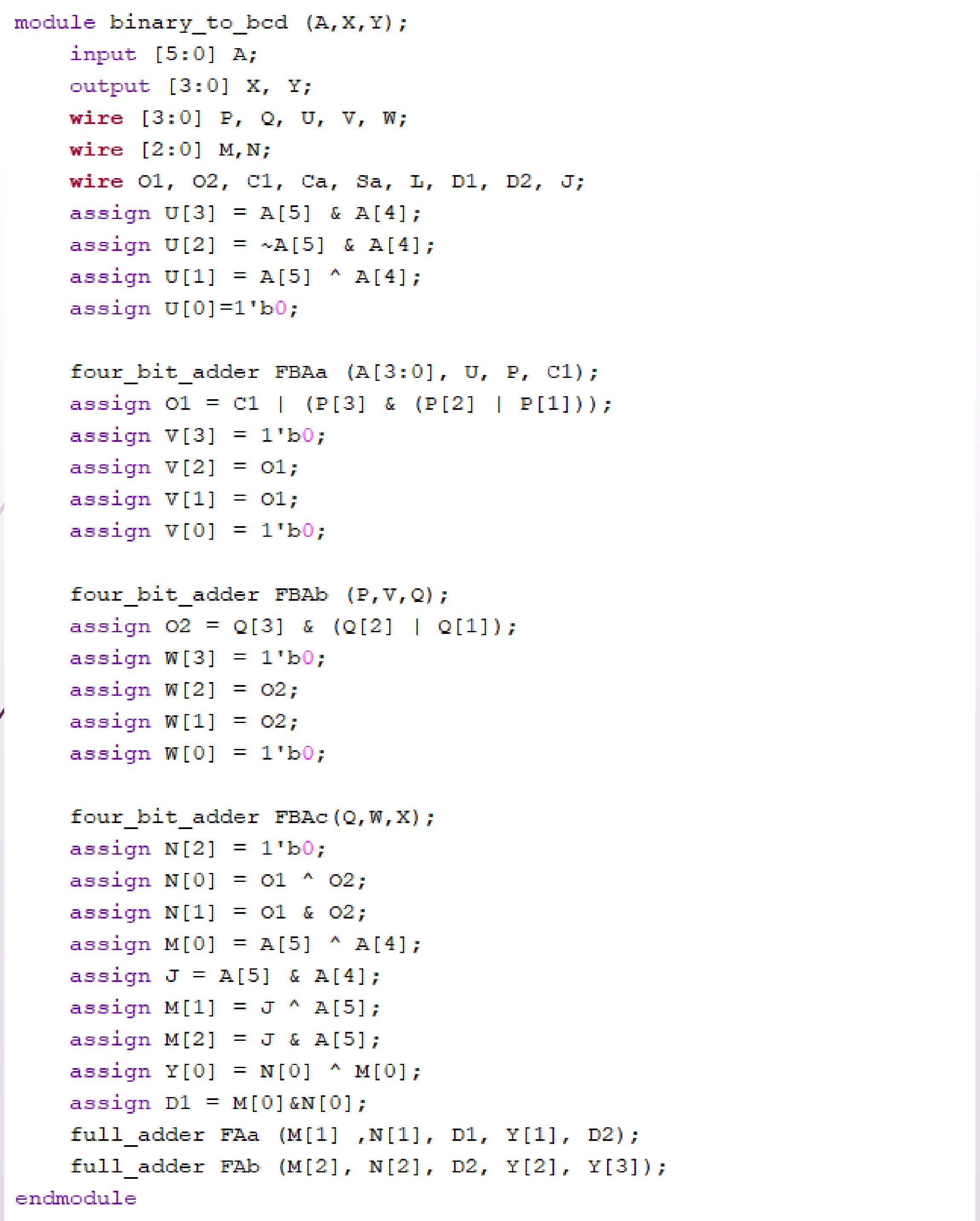
		A2	A1	A0
		B2	B1	B0
		A2B0	A1B0	A0B0
	A2B1	A1B1	A0B1	X
A2B2	A1B2	A0B2	X	X
A2B2+C+C	A2B1+A1B2 +A2B2+C+C	A1B1+C+ A0B2+A2B0	A0B1+A1B0	A0B0

Adding A2B0 and A1B1 will give rise to one carry, adding the sum obtained from that, and the carry obtained from adding A1B0 and A0B1 to A0B2 will give rise to another carry. Thus, two carries are generated and are carried over to the addition between A2B1 and A1B2, where two more carries are created similarly.
Hence the resulting circuit will contain nine AND gates, three half adders, and three full adders.

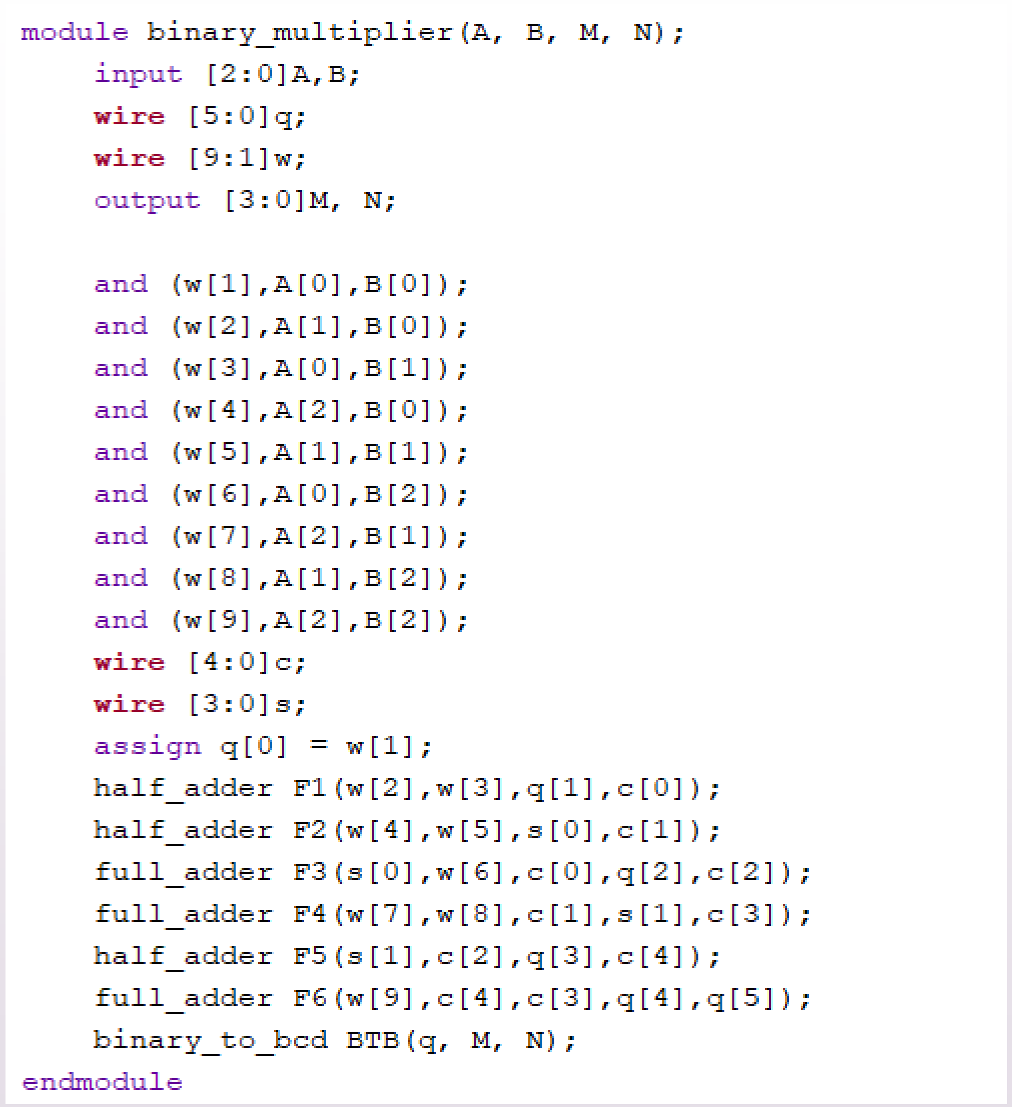
CIRCUIT DIAGRAM



CODE

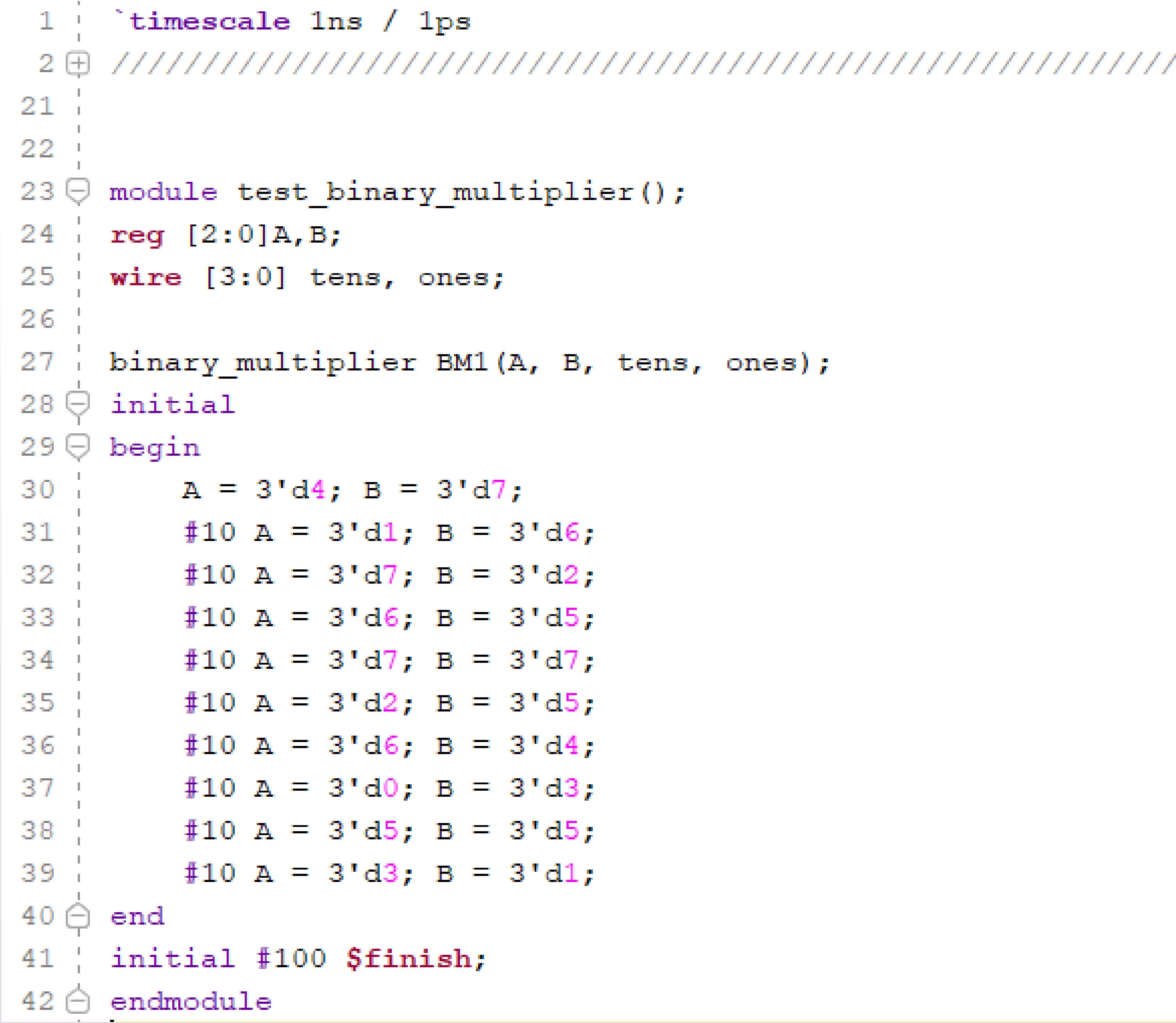


```
module binary_to_bcd (A,X,Y);
    input [5:0] A;
    output [3:0] X, Y;
    wire [3:0] P, Q, U, V, W;
    wire [2:0] M,N;
    wire O1, O2, C1, Ca, Sa, L, D1, D2, J;
    assign U[3] = A[5] & A[4];
    assign U[2] = ~A[5] & A[4];
    assign U[1] = A[5] ^ A[4];
    assign U[0]=1'b0;

    four_bit_adder FBAa (A[3:0], U, P, C1);
    assign O1 = C1 | (P[3] & (P[2] | P[1]));
    assign V[3] = 1'b0;
    assign V[2] = O1;
    assign V[1] = O1;
    assign V[0] = 1'b0;

    four_bit_adder FBAb (P,V,Q);
    assign O2 = Q[3] & (Q[2] | Q[1]);
    assign W[3] = 1'b0;
    assign W[2] = O2;
    assign W[1] = O2;
    assign W[0] = 1'b0;

    four_bit_adder FBAc(Q,W,X);
    assign N[2] = 1'b0;
    assign N[0] = O1 ^ O2;
    assign N[1] = O1 & O2;
    assign M[0] = A[5] ^ A[4];
    assign J = A[5] & A[4];
    assign M[1] = J ^ A[5];
    assign M[2] = J & A[5];
    assign Y[0] = N[0] ^ M[0];
    assign D1 = M[0]&N[0];
    full_adder FAa (M[1] ,N[1], D1, Y[1], D2);
    full_adder FAb (M[2], N[2], D2, Y[2], Y[3]);
endmodule
```

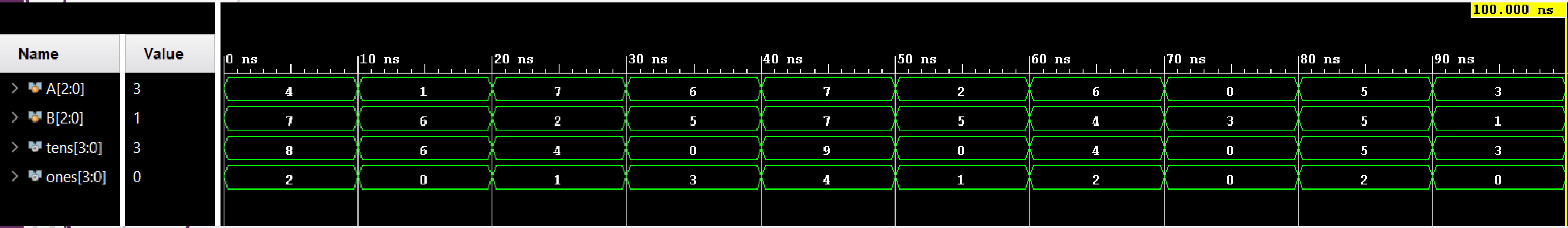


```
module binary_multiplier(A, B, M, N);
    input [2:0]A,B;
    wire [5:0]q;
    wire [9:1]w;
    output [3:0]M, N;

    and (w[1],A[0],B[0]);
    and (w[2],A[1],B[0]);
    and (w[3],A[0],B[1]);
    and (w[4],A[2],B[0]);
    and (w[5],A[1],B[1]);
    and (w[6],A[0],B[2]);
    and (w[7],A[2],B[1]);
    and (w[8],A[1],B[2]);
    and (w[9],A[2],B[2]);
    wire [4:0]c;
    wire [3:0]s;
    assign q[0] = w[1];
    half_adder F1(w[2],w[3],q[1],c[0]);
    half_adder F2(w[4],w[5],s[0],c[1]);
    full_adder F3(s[0],w[6],c[0],q[2],c[2]);
    full_adder F4(w[7],w[8],c[1],s[1],c[3]);
    half_adder F5(s[1],c[2],q[3],c[4]);
    full_adder F6(w[9],c[4],c[3],q[4],q[5]);
    binary_to_bcd BTB(q, M, N);
endmodule
```

TESTBENCH



```
`timescale 1ns / 1ps
/////////////////////////////////////////////////////////////////////////////////////////////////


module test_binary_multiplier();
reg [2:0]A,B;
wire [3:0] tens, ones;

binary_multiplier BM1(A, B, tens, ones);
initial
begin
    A = 3'd4; B = 3'd7;
    #10 A = 3'd1; B = 3'd6;
    #10 A = 3'd7; B = 3'd2;
    #10 A = 3'd6; B = 3'd5;
    #10 A = 3'd7; B = 3'd7;
    #10 A = 3'd2; B = 3'd5;
    #10 A = 3'd6; B = 3'd4;
    #10 A = 3'd0; B = 3'd3;
    #10 A = 3'd5; B = 3'd5;
    #10 A = 3'd3; B = 3'd1;
end
initial #100 $finish;
endmodule
```



100.000 ns

Name	Value	0 ns	10 ns	20 ns	30 ns	40 ns	50 ns	60 ns	70 ns	80 ns	90 ns
A[2:0]	3	4	1	7	6	7	2	6	0	5	3
B[2:0]	1	7	6	2	5	7	5	4	3	5	1
tens[3:0]	3	8	6	4	0	9	0	4	0	5	3
ones[3:0]	0	2	0	1	3	4	1	2	0	2	0

A = 4	A = 1	A = 7	A = 6	A = 7	A = 2	A = 6	A = 0	A = 5	A = 3
B = 7	B = 6	B = 2	B = 5	B = 7	B = 5	B = 4	B = 3	B = 5	B = 1
P = 28	P = 06	P = 14	P = 30	P = 49	P = 10	P = 24	P = 00	P = 25	P = 03

THANK YOU