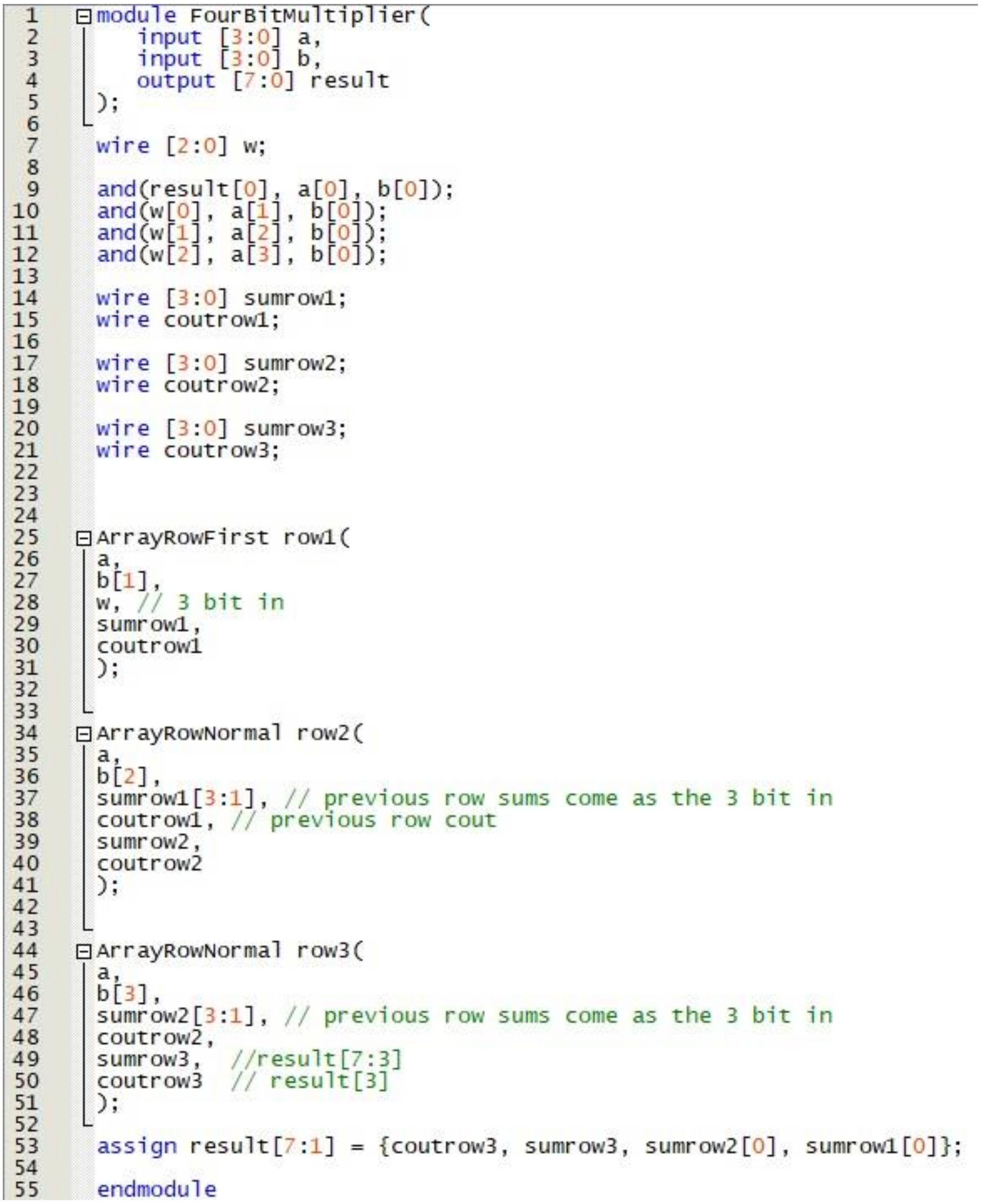
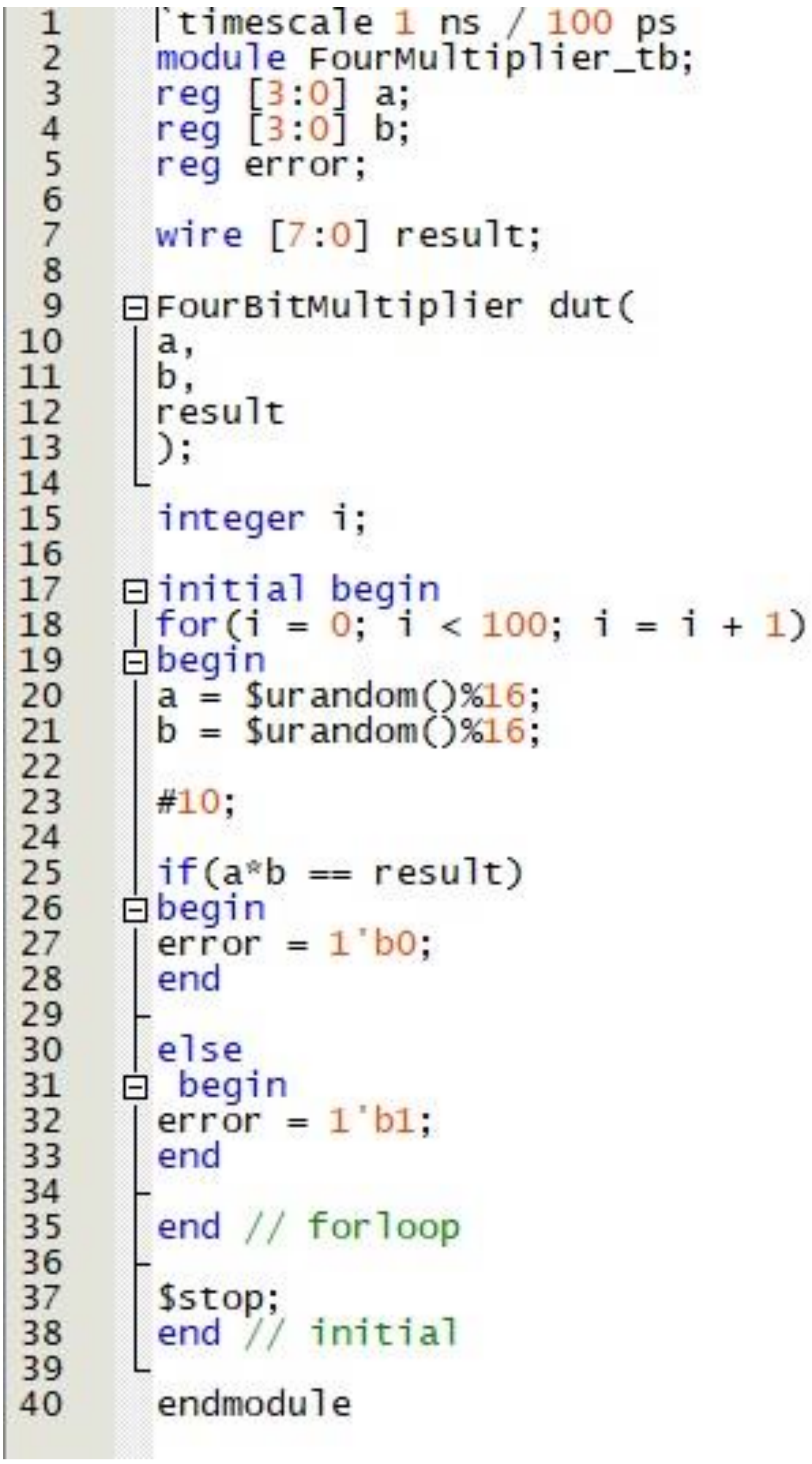
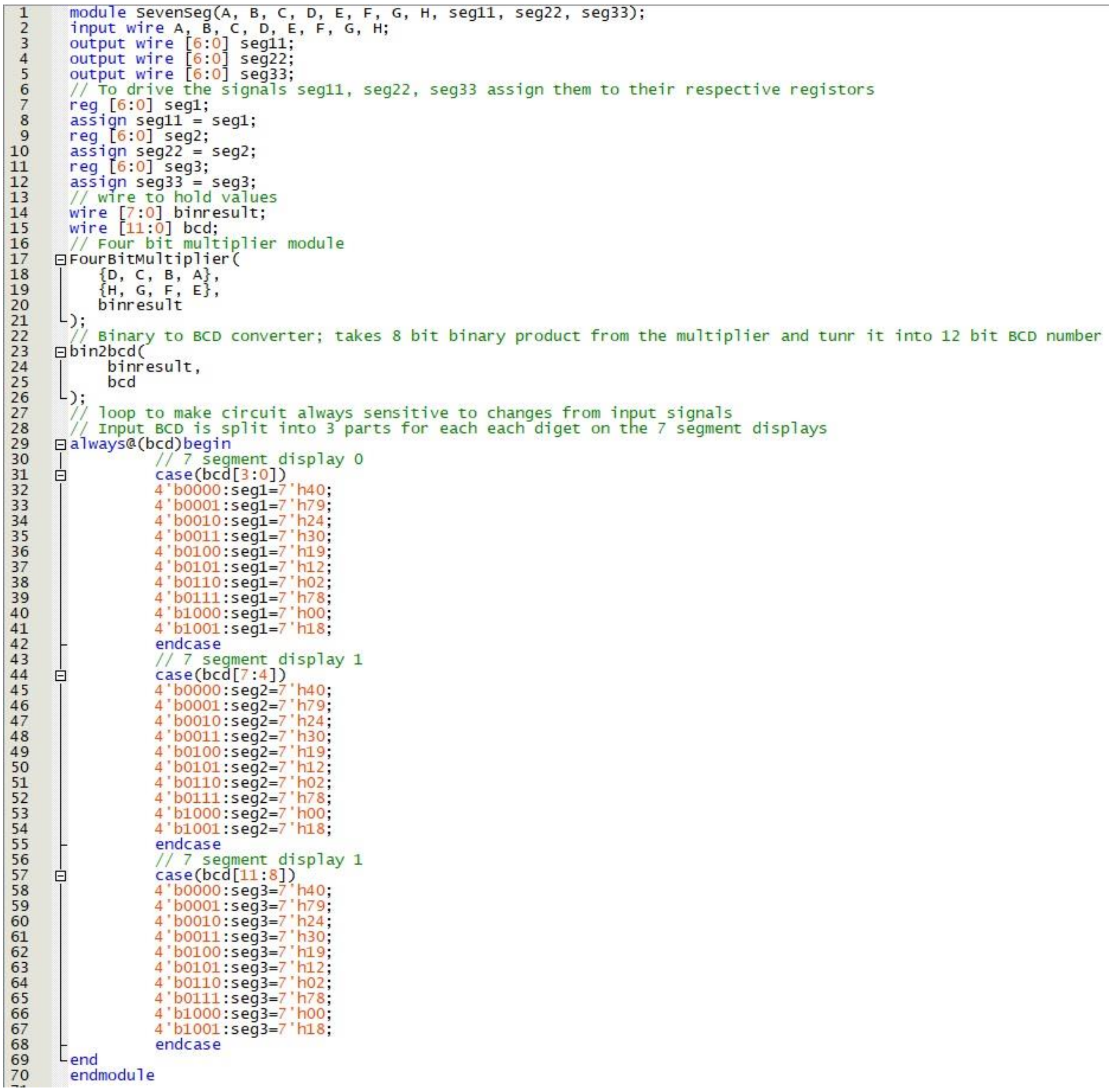


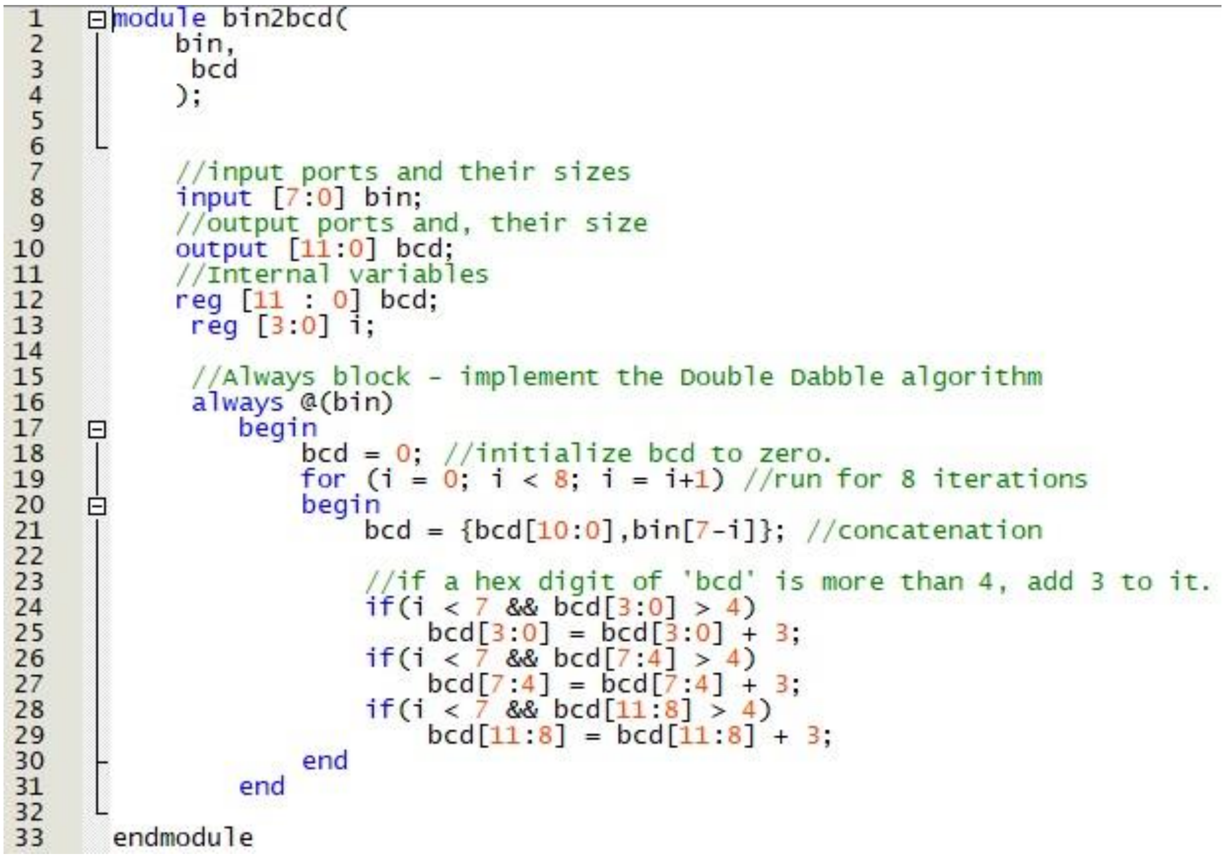
```verilog
module FourBitMultiplier(
    input [3:0] a,
    input [3:0] b,
    output [7:0] result
);

wire [2:0] w;

and(result[0], a[0], b[0]);
and(w[0], a[1], b[0]);
and(w[1], a[2], b[0]);
and(w[2], a[3], b[0]);

wire [3:0] sumrow1;
wire coutrow1;

wire [3:0] sumrow2;
wire coutrow2;

wire [3:0] sumrow3;
wire coutrow3;



ArrayRowFirst row1(
a,
b[1],
w, // 3 bit in
sumrow1,
coutrow1
);


ArrayRowNormal row2(
a,
b[2],
sumrow1[3:1], // previous row sums come as the 3 bit in
coutrow1, // previous row cout
sumrow2,
coutrow2
);


ArrayRowNormal row3(
a,
b[3],
sumrow2[3:1], // previous row sums come as the 3 bit in
coutrow2,
sumrow3,  //result[7:3]
coutrow3  // result[3]
);

assign result[7:1] = {coutrow3, sumrow3, sumrow2[0], sumrow1[0]};

endmodule
```



```
`timescale 1 ns / 100 ps
module FourMultiplier_tb;
reg [3:0] a;
reg [3:0] b;
reg error;

wire [7:0] result;

FourBitMultiplier dut(
a,
b,
result
);

integer i;

initial begin
for(i = 0; i < 100; i = i + 1)
begin
a = $urandom()%16;
b = $urandom()%16;

#10;

if(a*b == result)
begin
error = 1'b0;
end

else
 begin
error = 1'b1;
end

end // forloop

$stop;
end // initial

endmodule
```



```verilog
module SevenSeg(A, B, C, D, E, F, G, H, seg11, seg22, seg33);
input wire A, B, C, D, E, F, G, H;
output wire [6:0] seg11;
output wire [6:0] seg22;
output wire [6:0] seg33;
// To drive the signals seg11, seg22, seg33 assign them to their respective registors
reg [6:0] seg1;
assign seg11 = seg1;
reg [6:0] seg2;
assign seg22 = seg2;
reg [6:0] seg3;
assign seg33 = seg3;
// wire to hold values
wire [7:0] binresult;
wire [11:0] bcd;
// Four bit multiplier module
FourBitMultiplier(
   {D, C, B, A},
   {H, G, F, E},
   binresult
);
// Binary to BCD converter; takes 8 bit binary product from the multiplier and tunr it into 12 bit BCD number
bin2bcd(
    binresult,
    bcd
);
// loop to make circuit always sensitive to changes from input signals
// Input BCD is split into 3 parts for each each diget on the 7 segment displays
always@(bcd)begin
         // 7 segment display 0
         case(bcd[3:0])
         4'b0000:seg1=7'h40;
         4'b0001:seg1=7'h79;
         4'b0010:seg1=7'h24;
         4'b0011:seg1=7'h30;
         4'b0100:seg1=7'h19;
         4'b0101:seg1=7'h12;
         4'b0110:seg1=7'h02;
         4'b0111:seg1=7'h78;
         4'b1000:seg1=7'h00;
         4'b1001:seg1=7'h18;
         endcase
         // 7 segment display 1
         case(bcd[7:4])
         4'b0000:seg2=7'h40;
         4'b0001:seg2=7'h79;
         4'b0010:seg2=7'h24;
         4'b0011:seg2=7'h30;
         4'b0100:seg2=7'h19;
         4'b0101:seg2=7'h12;
         4'b0110:seg2=7'h02;
         4'b0111:seg2=7'h78;
         4'b1000:seg2=7'h00;
         4'b1001:seg2=7'h18;
         endcase
         // 7 segment display 1
         case(bcd[11:8])
         4'b0000:seg3=7'h40;
         4'b0001:seg3=7'h79;
         4'b0010:seg3=7'h24;
         4'b0011:seg3=7'h30;
         4'b0100:seg3=7'h19;
         4'b0101:seg3=7'h12;
         4'b0110:seg3=7'h02;
         4'b0111:seg3=7'h78;
         4'b1000:seg3=7'h00;
         4'b1001:seg3=7'h18;
         endcase
end
endmodule
```



```verilog
module bin2bcd(
    bin,
     bcd
    );


    //input ports and their sizes
    input [7:0] bin;
    //output ports and, their size
    output [11:0] bcd;
    //Internal variables
    reg [11 : 0] bcd;
     reg [3:0] i;

     //Always block - implement the Double Dabble algorithm
     always @(bin)
        begin
            bcd = 0; //initialize bcd to zero.
            for (i = 0; i < 8; i = i+1) //run for 8 iterations
            begin
                bcd = {bcd[10:0],bin[7-i]}; //concatenation

                //if a hex digit of 'bcd' is more than 4, add 3 to it.
                if(i < 7 && bcd[3:0] > 4)
                    bcd[3:0] = bcd[3:0] + 3;
                if(i < 7 && bcd[7:4] > 4)
                    bcd[7:4] = bcd[7:4] + 3;
                if(i < 7 && bcd[11:8] > 4)
                    bcd[11:8] = bcd[11:8] + 3;
            end
        end

endmodule
```