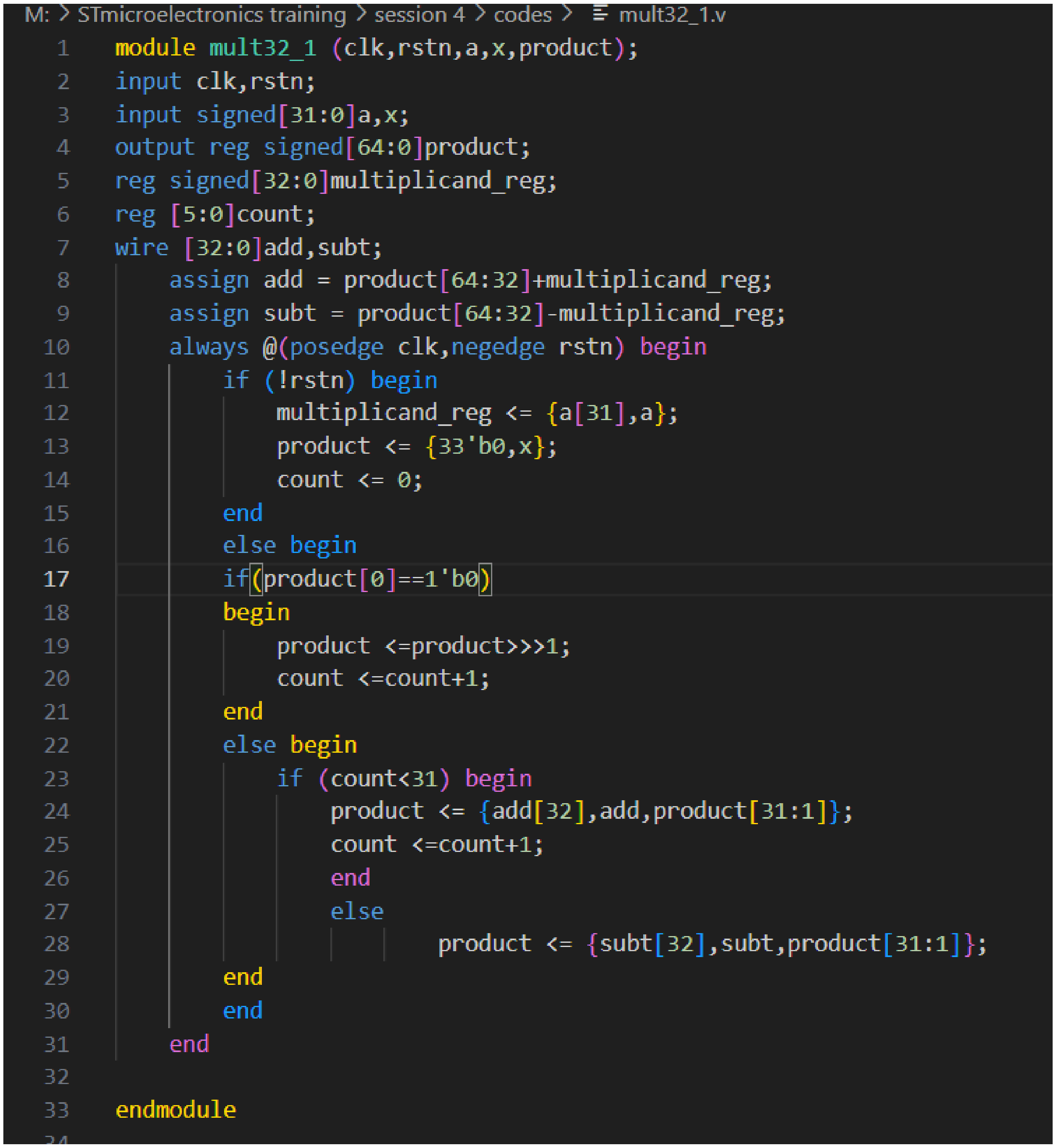
Momen Mostafa Mohamed Elzaghawy

compArith_lab3

Q1

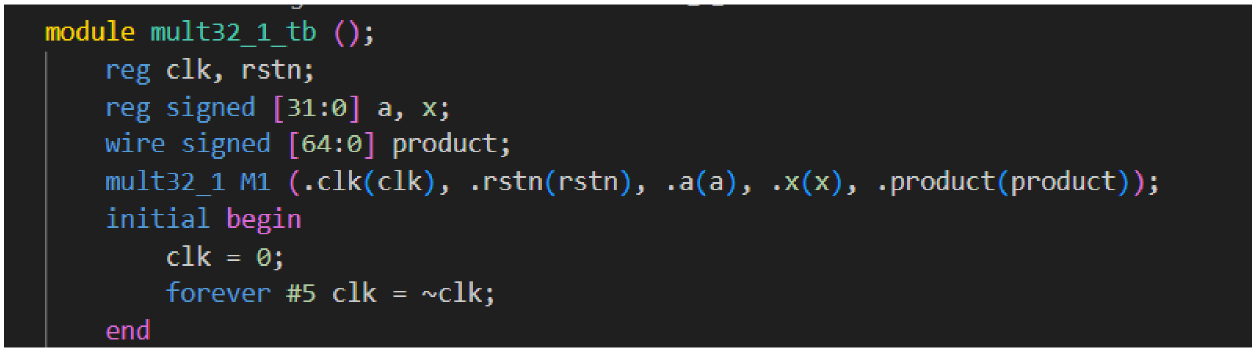
RTL:



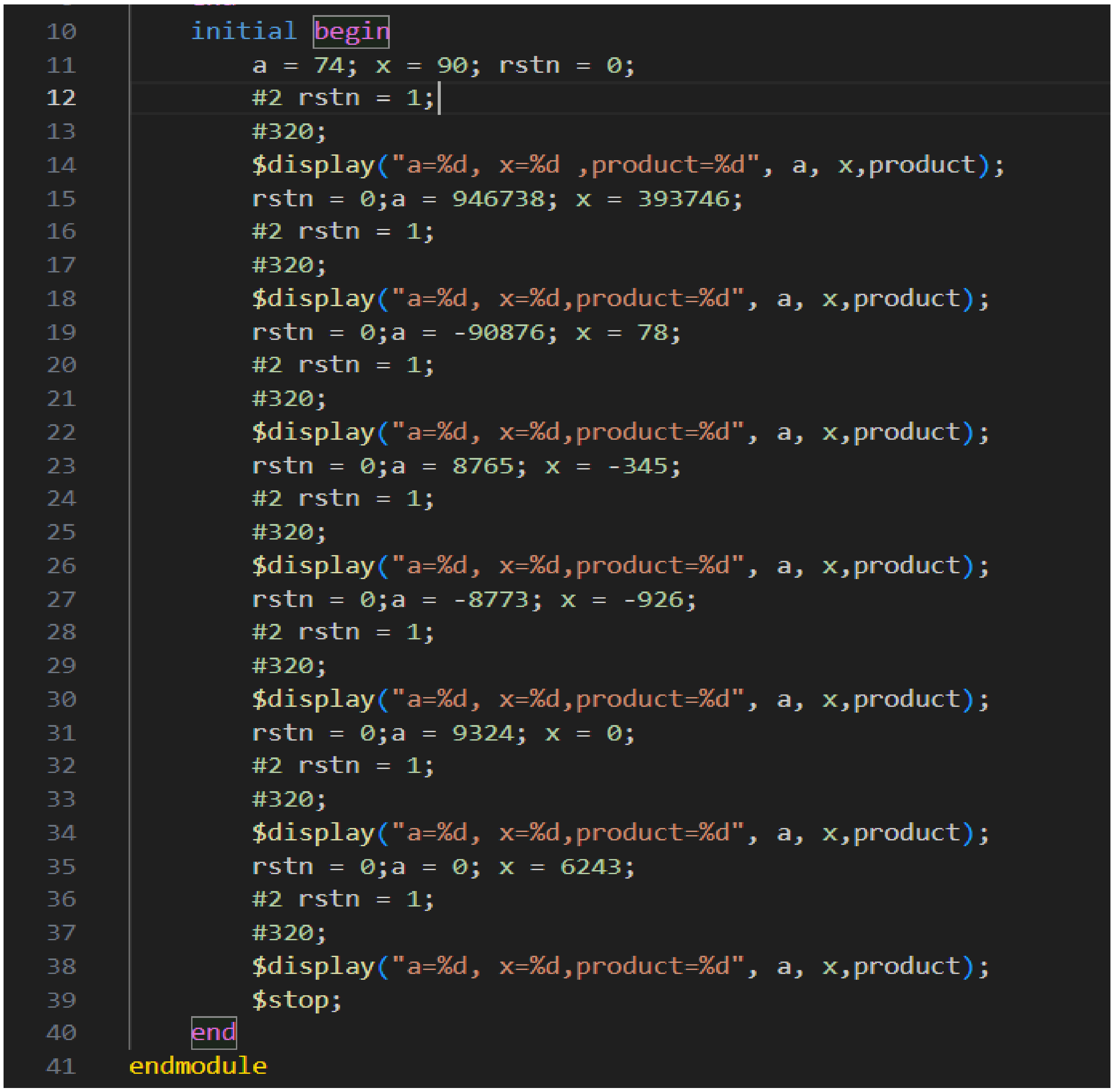
M: > STmicroelectronics training > session 4 > codes > mult32_1.v

```
module mult32_1 (clk,rstn,a,x,product);
input clk,rstn;
input signed[31:0]a,x;
output reg signed[64:0]product;
reg signed[32:0]multiplicand_reg;
reg [5:0]count;
wire [32:0]add,subt;
    assign add = product[64:32]+multiplicand_reg;
    assign subt = product[64:32]-multiplicand_reg;
    always @(posedge clk,negedge rstn) begin
        if (!rstn) begin
            multiplicand_reg <= {a[31],a};
            product <= {33'b0,x};
            count <= 0;
        end
        else begin
        if(product[0]==1'b0)
        begin
            product <=product>>>1;
            count <=count+1;
        end
        else begin
            if (count<31) begin
                product <= {add[32],add,product[31:1]};
                count <=count+1;
                end
                else
                        product <= {subt[32],subt,product[31:1]};
        end
        end
    end

endmodule
```

Testbench:

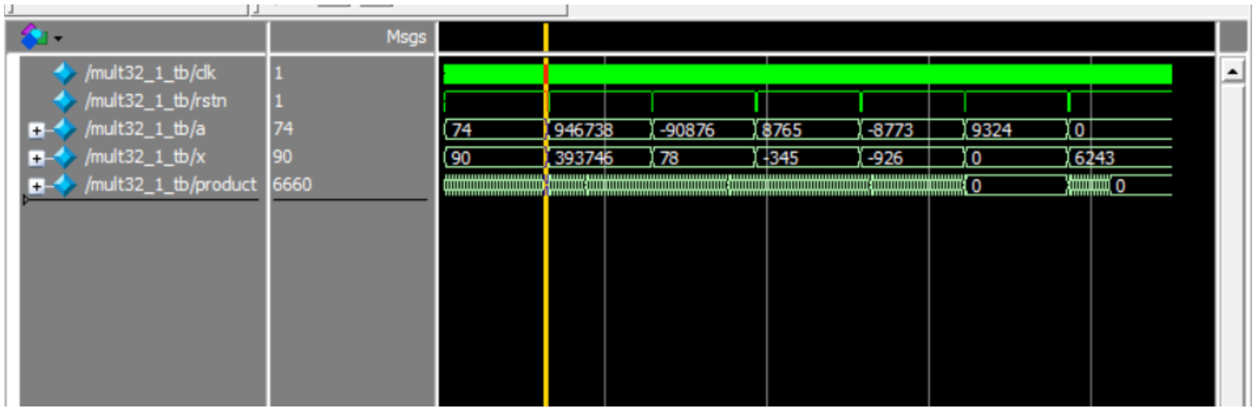


```verilog
module mult32_1_tb ();
    reg clk, rstn;
    reg signed [31:0] a, x;
    wire signed [64:0] product;
    mult32_1 M1 (.clk(clk), .rstn(rstn), .a(a), .x(x), .product(product));
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
```

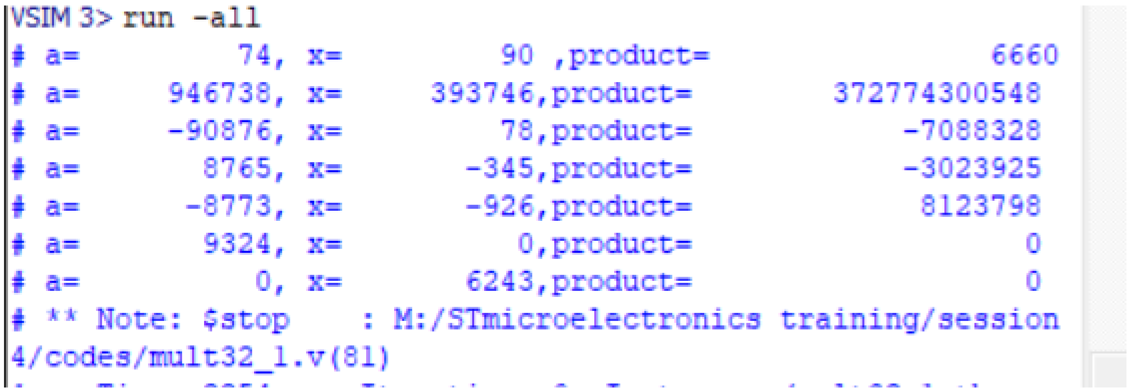


```verilog
    initial begin
        a = 74; x = 90; rstn = 0;
        #2 rstn = 1;
        #320;
        $display("a=%d, x=%d ,product=%d", a, x,product);
        rstn = 0;a = 946738; x = 393746;
        #2 rstn = 1;
        #320;
        $display("a=%d, x=%d,product=%d", a, x,product);
        rstn = 0;a = -90876; x = 78;
        #2 rstn = 1;
        #320;
        $display("a=%d, x=%d,product=%d", a, x,product);
        rstn = 0;a = 8765; x = -345;
        #2 rstn = 1;
        #320;
        $display("a=%d, x=%d,product=%d", a, x,product);
        rstn = 0;a = -8773; x = -926;
        #2 rstn = 1;
        #320;
        $display("a=%d, x=%d,product=%d", a, x,product);
        rstn = 0;a = 9324; x = 0;
        #2 rstn = 1;
        #320;
        $display("a=%d, x=%d,product=%d", a, x,product);
        rstn = 0;a = 0; x = 6243;
        #2 rstn = 1;
        #320;
        $display("a=%d, x=%d,product=%d", a, x,product);
        $stop;
    end
endmodule
```

Wave form, transcript:

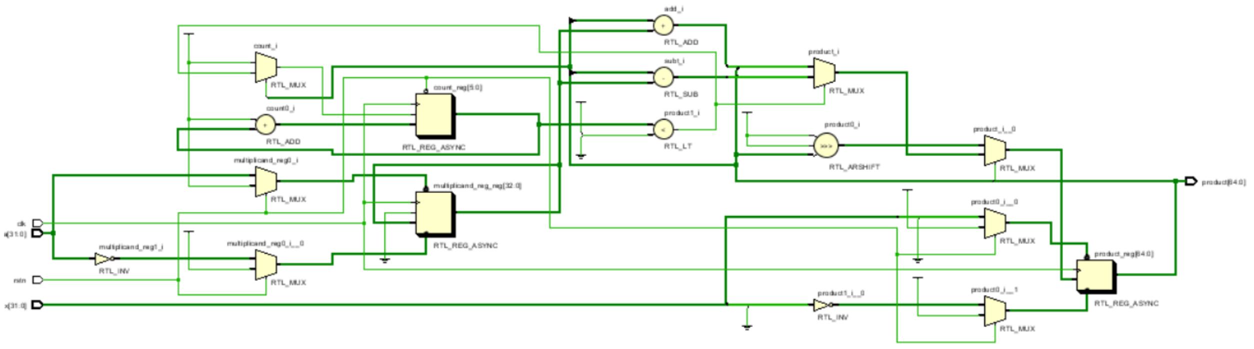


```
Msgs
/mult32_1_tb/clk      1
/mult32_1_tb/rstn     1
/mult32_1_tb/a        74     74    946738  -90876  8765   -8773  9324  0
/mult32_1_tb/x        90     90    393746  78      -345   -926   0     6243
/mult32_1_tb/product  6660                                       0          0
```

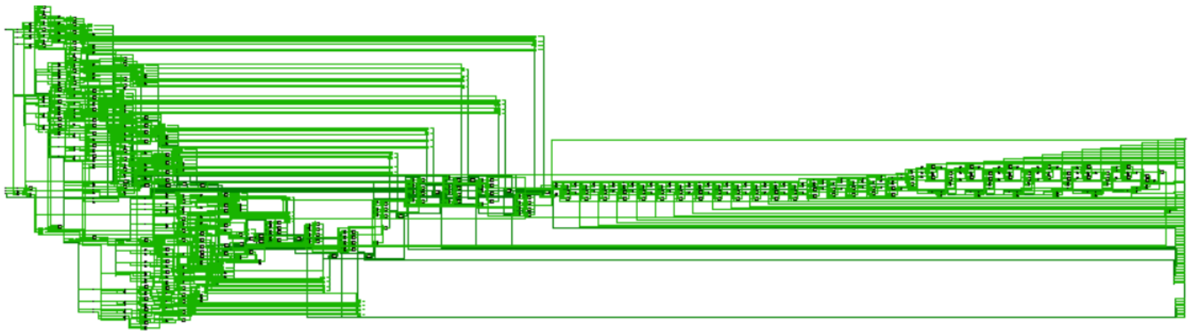


```
VSIM 3> run -all
# a=         74, x=         90 ,product=                6660
# a=     946738, x=     393746,product=         372774300548
# a=     -90876, x=         78,product=              -7088328
# a=       8765, x=       -345,product=              -3023925
# a=      -8773, x=       -926,product=               8123798
# a=       9324, x=          0,product=                     0
# a=          0, x=       6243,product=                     0
# ** Note: $stop    : M:/STmicroelectronics training/session
4/codes/mult32_1.v(81)
```

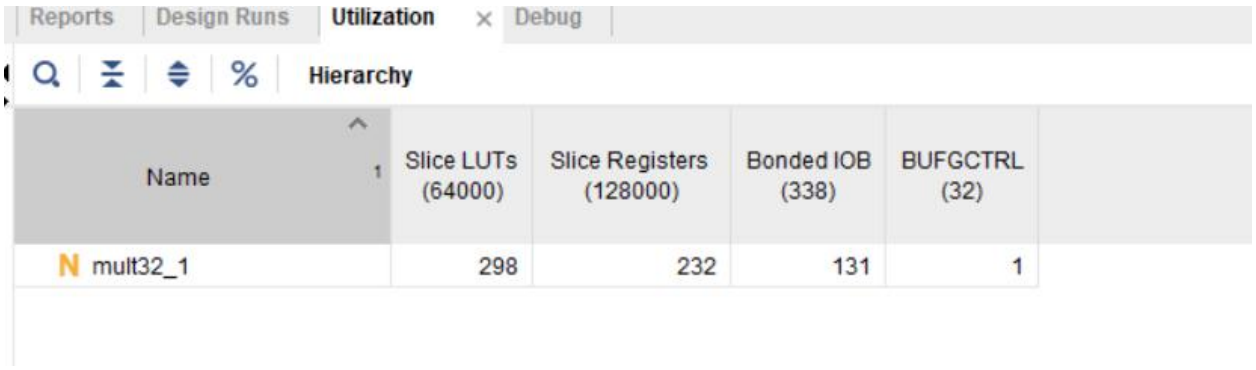
Schematic:



Synthesis:

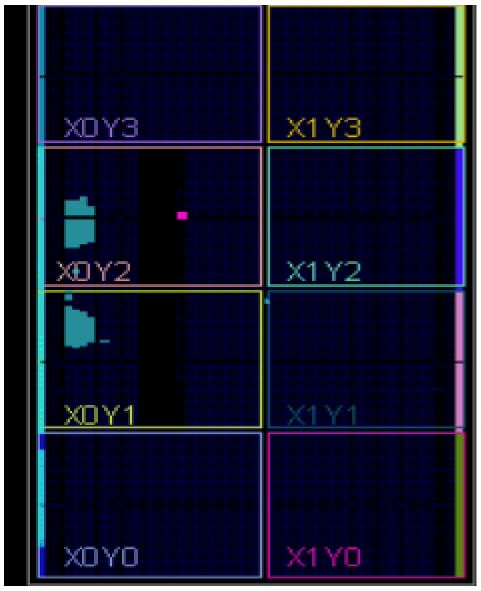


Utilization:



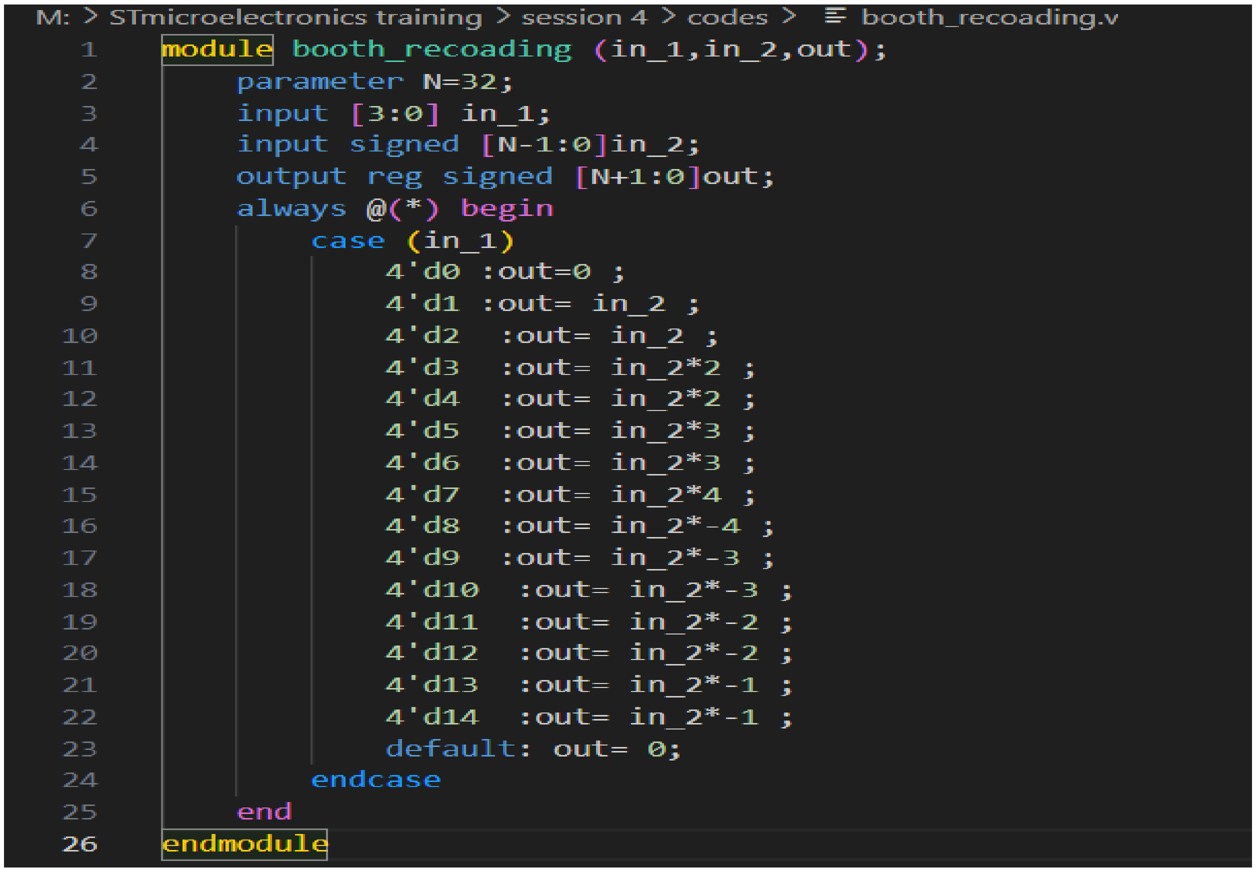
Name	Slice LUTs (64000)	Slice Registers (128000)	Bonded IOB (338)	BUFGCTRL (32)
mult32_1	298	232	131	1

Implementation:



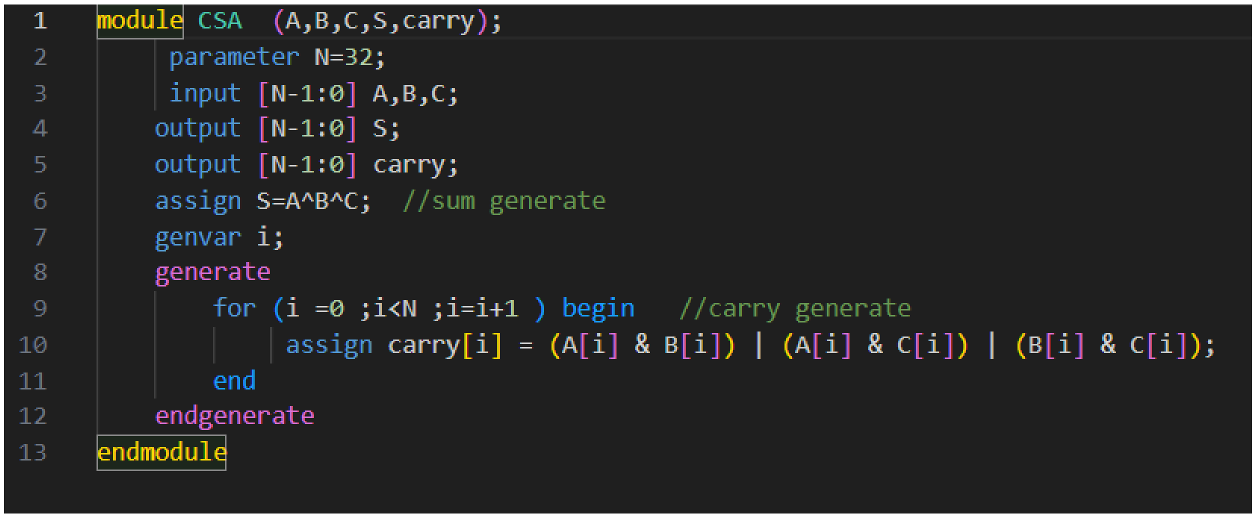
Q2

RTL:

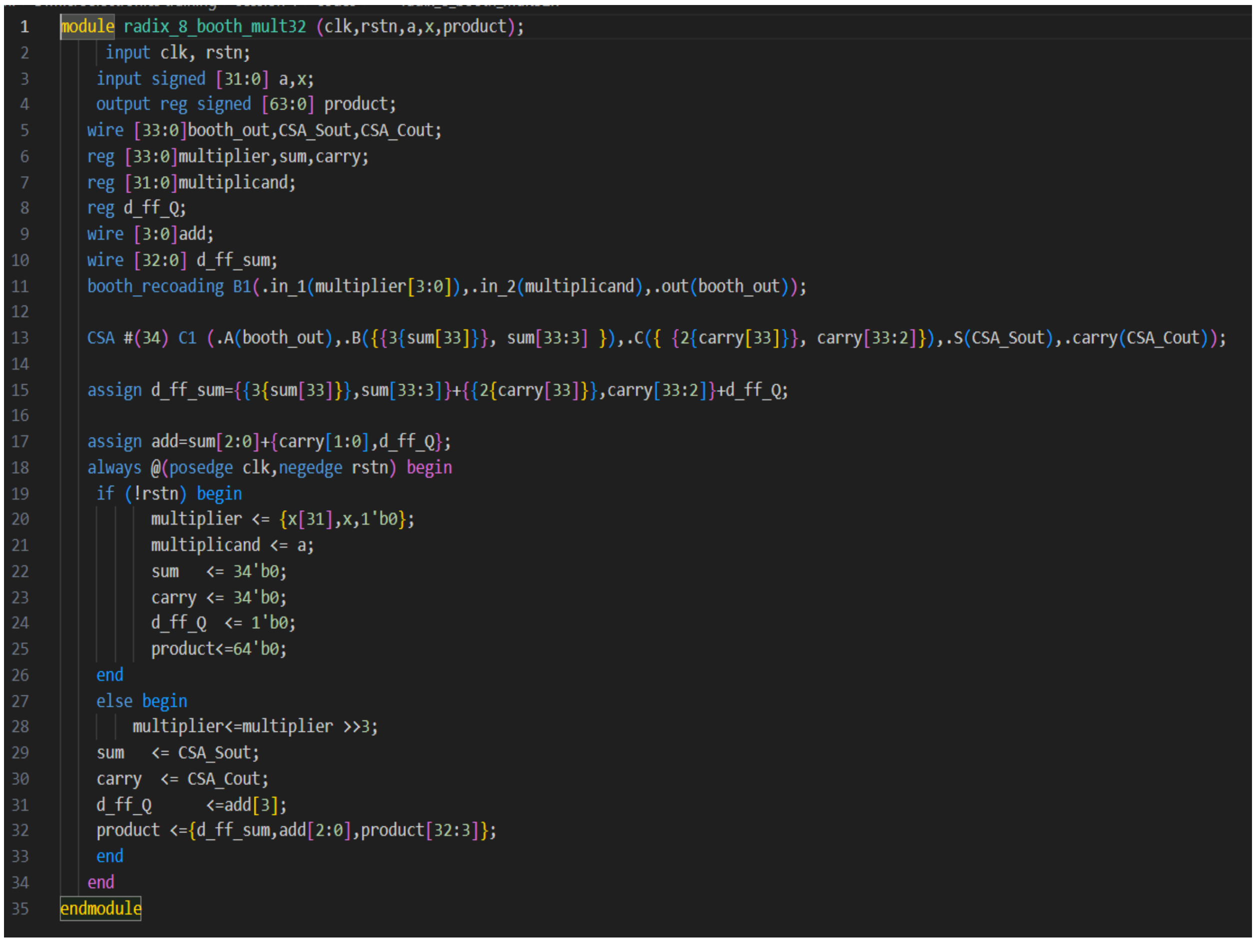


M: > STmicroelectronics training > session 4 > codes > booth_recoading.v

```
module booth_recoading (in_1,in_2,out);
    parameter N=32;
    input [3:0] in_1;
    input signed [N-1:0]in_2;
    output reg signed [N+1:0]out;
    always @(*) begin
        case (in_1)
            4'd0 :out=0 ;
            4'd1 :out= in_2 ;
            4'd2  :out= in_2 ;
            4'd3  :out= in_2*2 ;
            4'd4  :out= in_2*2 ;
            4'd5  :out= in_2*3 ;
            4'd6  :out= in_2*3 ;
            4'd7  :out= in_2*4 ;
            4'd8  :out= in_2*-4 ;
            4'd9  :out= in_2*-3 ;
            4'd10  :out= in_2*-3 ;
            4'd11  :out= in_2*-2 ;
            4'd12  :out= in_2*-2 ;
            4'd13  :out= in_2*-1 ;
            4'd14  :out= in_2*-1 ;
            default: out= 0;
        endcase
    end
endmodule
```

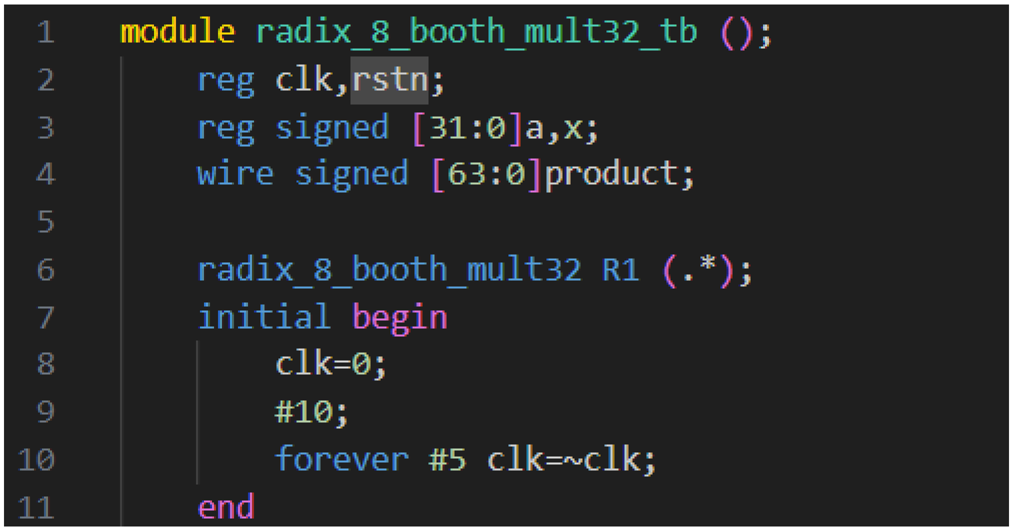


```
module CSA  (A,B,C,S,carry);
     parameter N=32;
     input [N-1:0] A,B,C;
    output [N-1:0] S;
    output [N-1:0] carry;
    assign S=A^B^C;  //sum generate
    genvar i;
    generate
        for (i =0 ;i<N ;i=i+1 ) begin   //carry generate
            assign carry[i] = (A[i] & B[i]) | (A[i] & C[i]) | (B[i] & C[i]);
        end
    endgenerate
endmodule
```

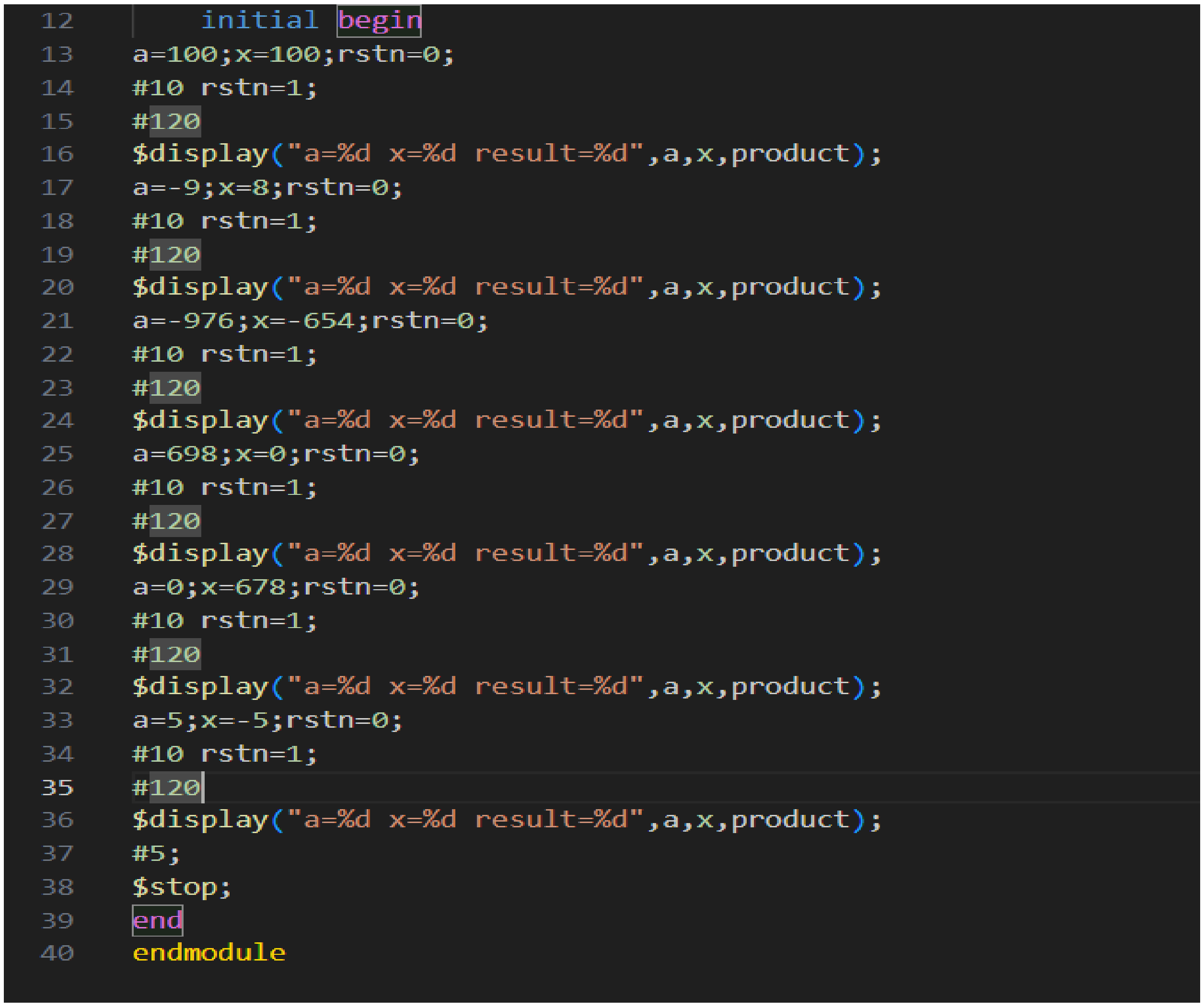


```verilog
module radix_8_booth_mult32 (clk,rstn,a,x,product);
     input clk, rstn;
    input signed [31:0] a,x;
    output reg signed [63:0] product;
   wire [33:0]booth_out,CSA_Sout,CSA_Cout;
   reg [33:0]multiplier,sum,carry;
   reg [31:0]multiplicand;
   reg d_ff_Q;
   wire [3:0]add;
   wire [32:0] d_ff_sum;
   booth_recoading B1(.in_1(multiplier[3:0]),.in_2(multiplicand),.out(booth_out));

   CSA #(34) C1 (.A(booth_out),.B({{3{sum[33]}}, sum[33:3] }),.C({ {2{carry[33]}}, carry[33:2]}),.S(CSA_Sout),.carry(CSA_Cout));

   assign d_ff_sum={{3{sum[33]}},sum[33:3]}+{{2{carry[33]}},carry[33:2]}+d_ff_Q;

   assign add=sum[2:0]+{carry[1:0],d_ff_Q};
   always @(posedge clk,negedge rstn) begin
    if (!rstn) begin
          multiplier <= {x[31],x,1'b0};
          multiplicand <= a;
          sum   <= 34'b0;
          carry <= 34'b0;
          d_ff_Q  <= 1'b0;
          product<=64'b0;
    end
    else begin
        multiplier<=multiplier >>3;
    sum   <= CSA_Sout;
    carry  <= CSA_Cout;
    d_ff_Q      <=add[3];
    product <={d_ff_sum,add[2:0],product[32:3]};
    end
   end
endmodule
```

Testbench:

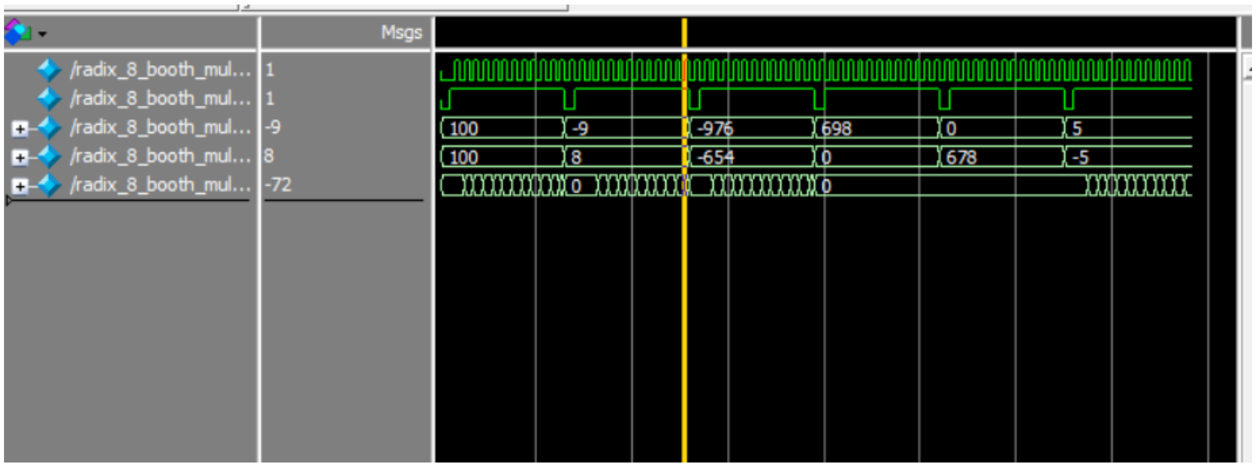


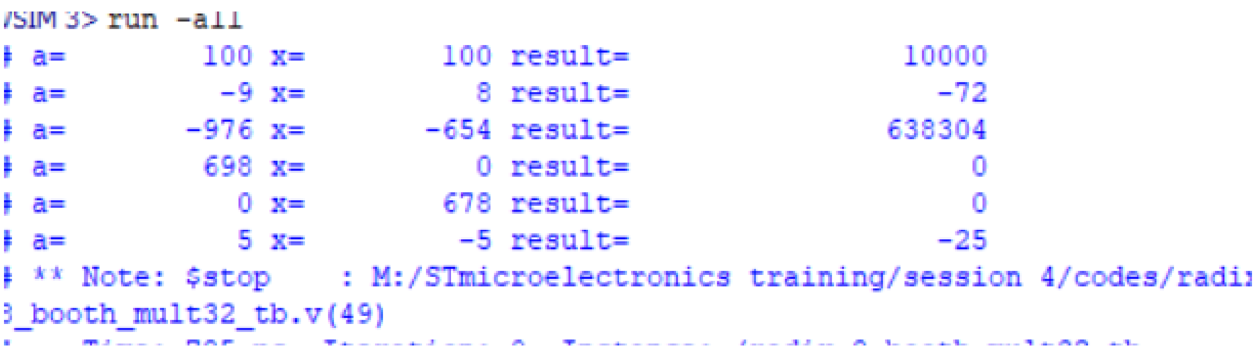
```
module radix_8_booth_mult32_tb ();
    reg clk,rstn;
    reg signed [31:0]a,x;
    wire signed [63:0]product;

    radix_8_booth_mult32 R1 (.*);
    initial begin
        clk=0;
        #10;
        forever #5 clk=~clk;
    end
```



```
    initial begin
a=100;x=100;rstn=0;
#10 rstn=1;
#120
$display("a=%d x=%d result=%d",a,x,product);
a=-9;x=8;rstn=0;
#10 rstn=1;
#120
$display("a=%d x=%d result=%d",a,x,product);
a=-976;x=-654;rstn=0;
#10 rstn=1;
#120
$display("a=%d x=%d result=%d",a,x,product);
a=698;x=0;rstn=0;
#10 rstn=1;
#120
$display("a=%d x=%d result=%d",a,x,product);
a=0;x=678;rstn=0;
#10 rstn=1;
#120
$display("a=%d x=%d result=%d",a,x,product);
a=5;x=-5;rstn=0;
#10 rstn=1;
#120
$display("a=%d x=%d result=%d",a,x,product);
#5;
$stop;
end
endmodule
```

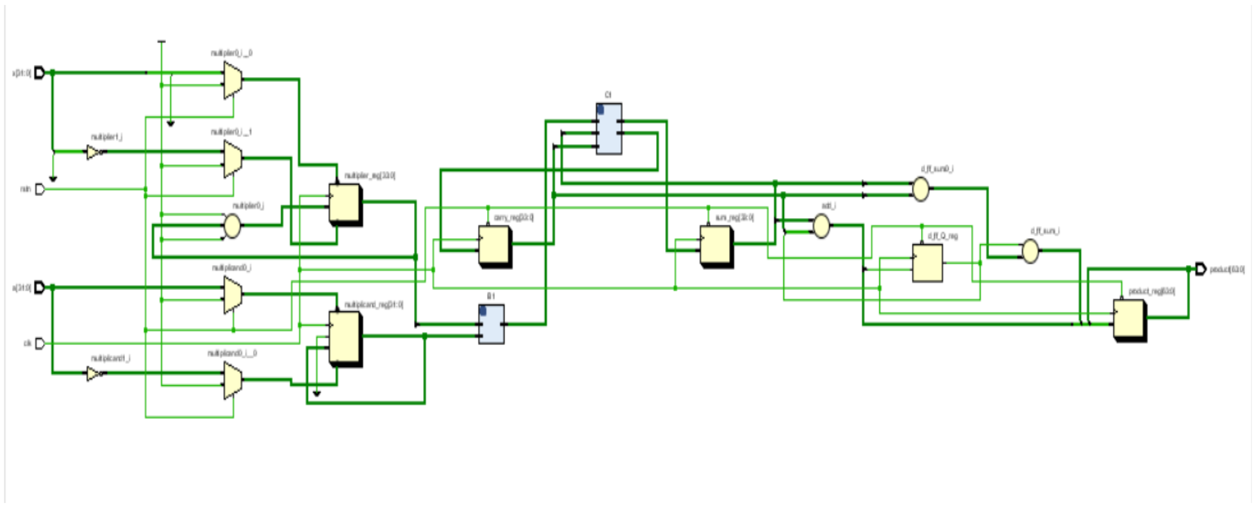
Wave &transcript:



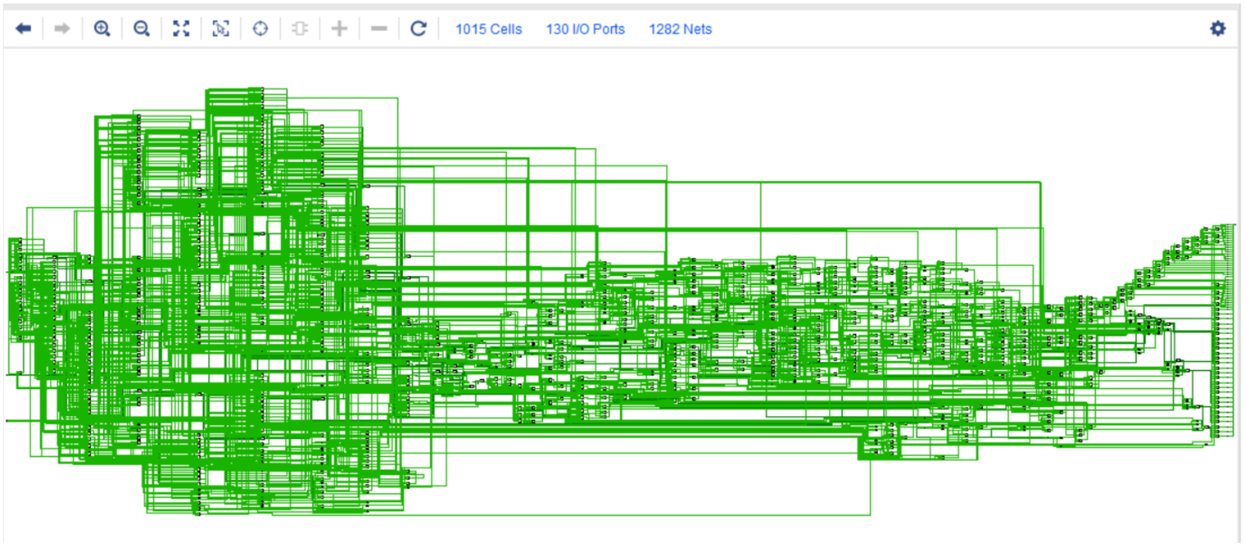


```
VSIM 3> run -all
# a=         100 x=         100 result=                 10000
# a=          -9 x=           8 result=                   -72
# a=        -976 x=        -654 result=                638304
# a=         698 x=           0 result=                     0
# a=           0 x=         678 result=                     0
# a=           5 x=          -5 result=                   -25
# ** Note: $stop    : M:/STmicroelectronics training/session 4/codes/radi
8_booth_mult32_tb.v(49)
```

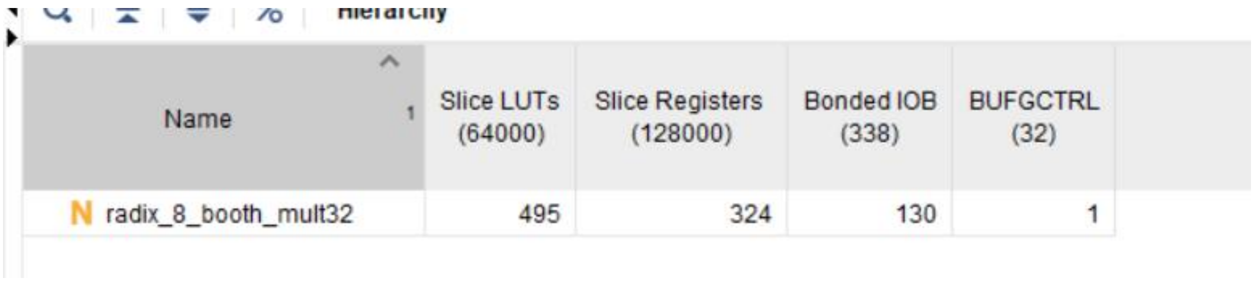
Schematic:



Synthesis:



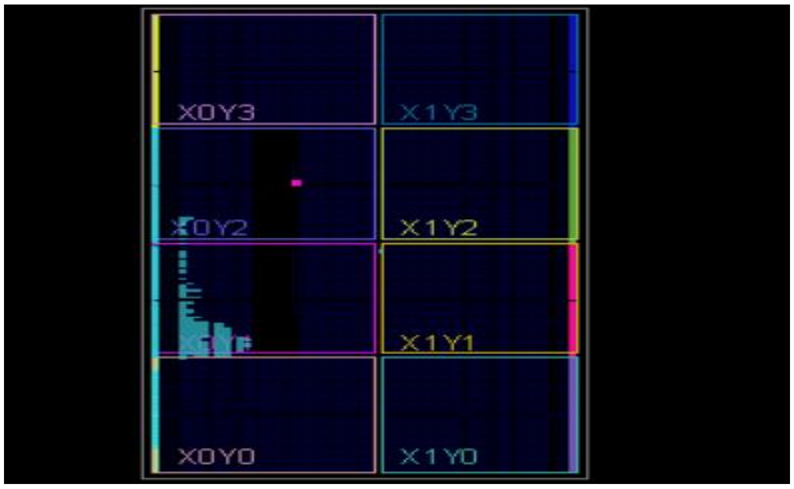
Utilization:



Hierarchy

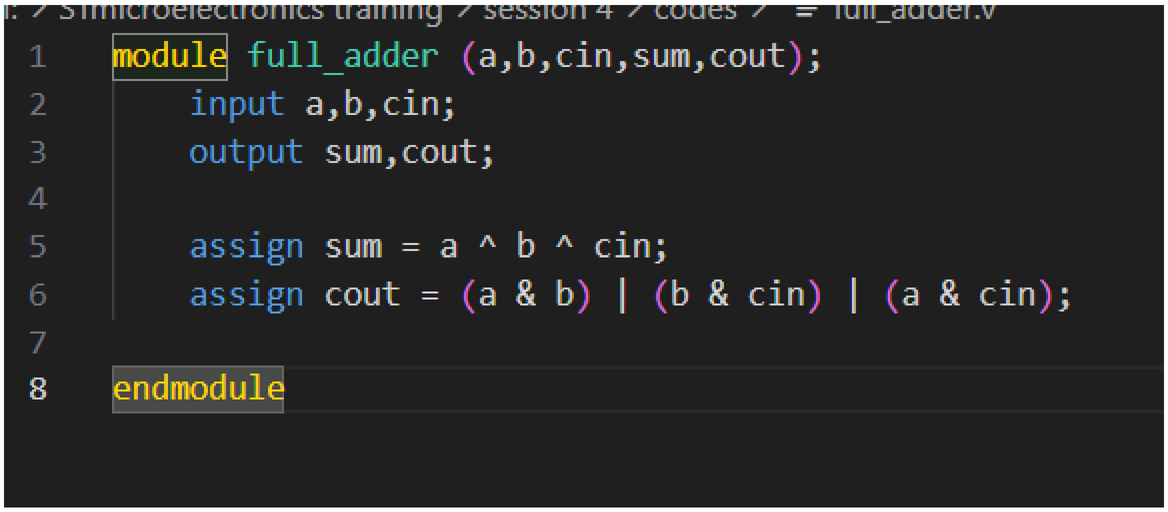
Name	Slice LUTs (64000)	Slice Registers (128000)	Bonded IOB (338)	BUFGCTRL (32)
radix_8_booth_mult32	495	324	130	1

Implementation:

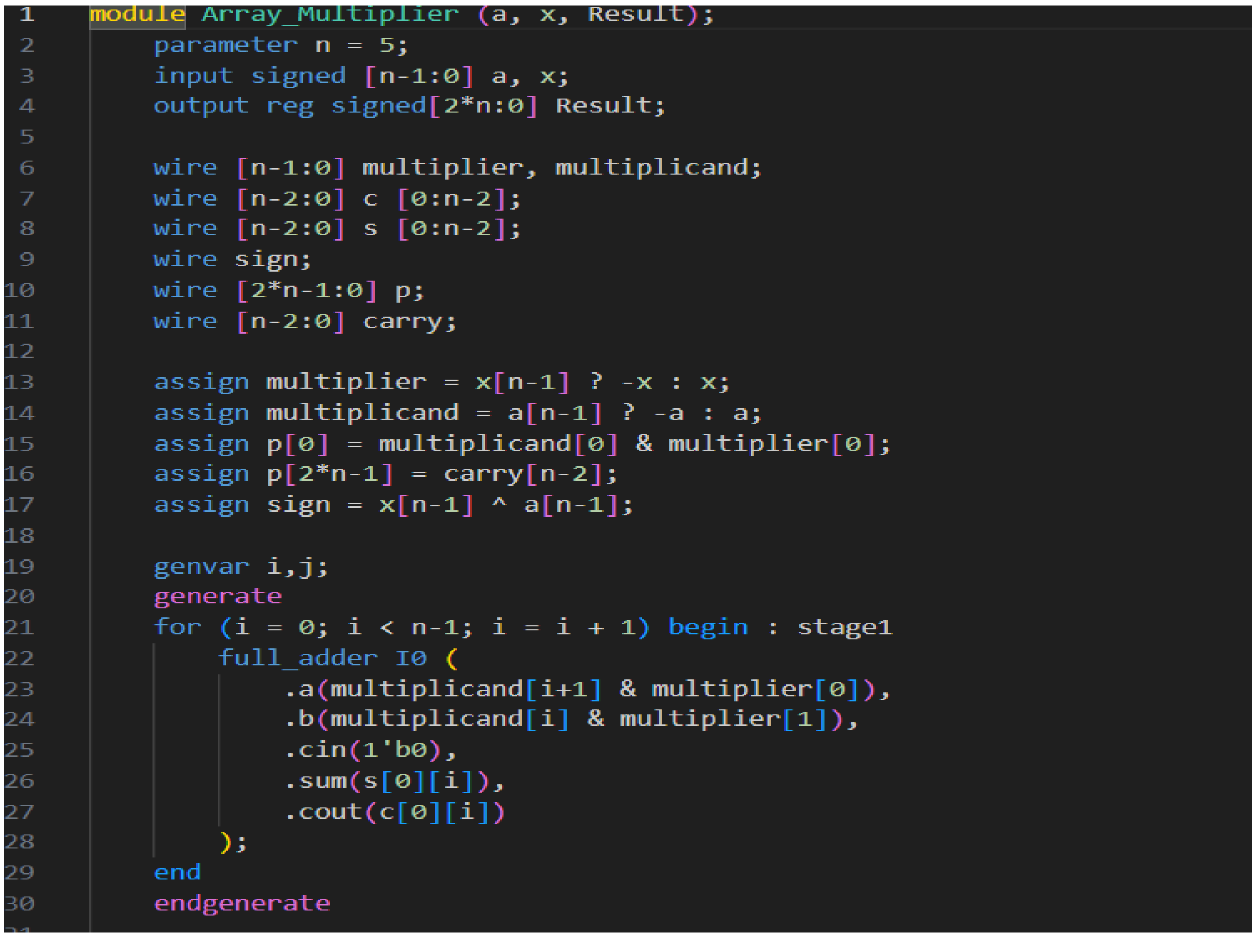


Q3:

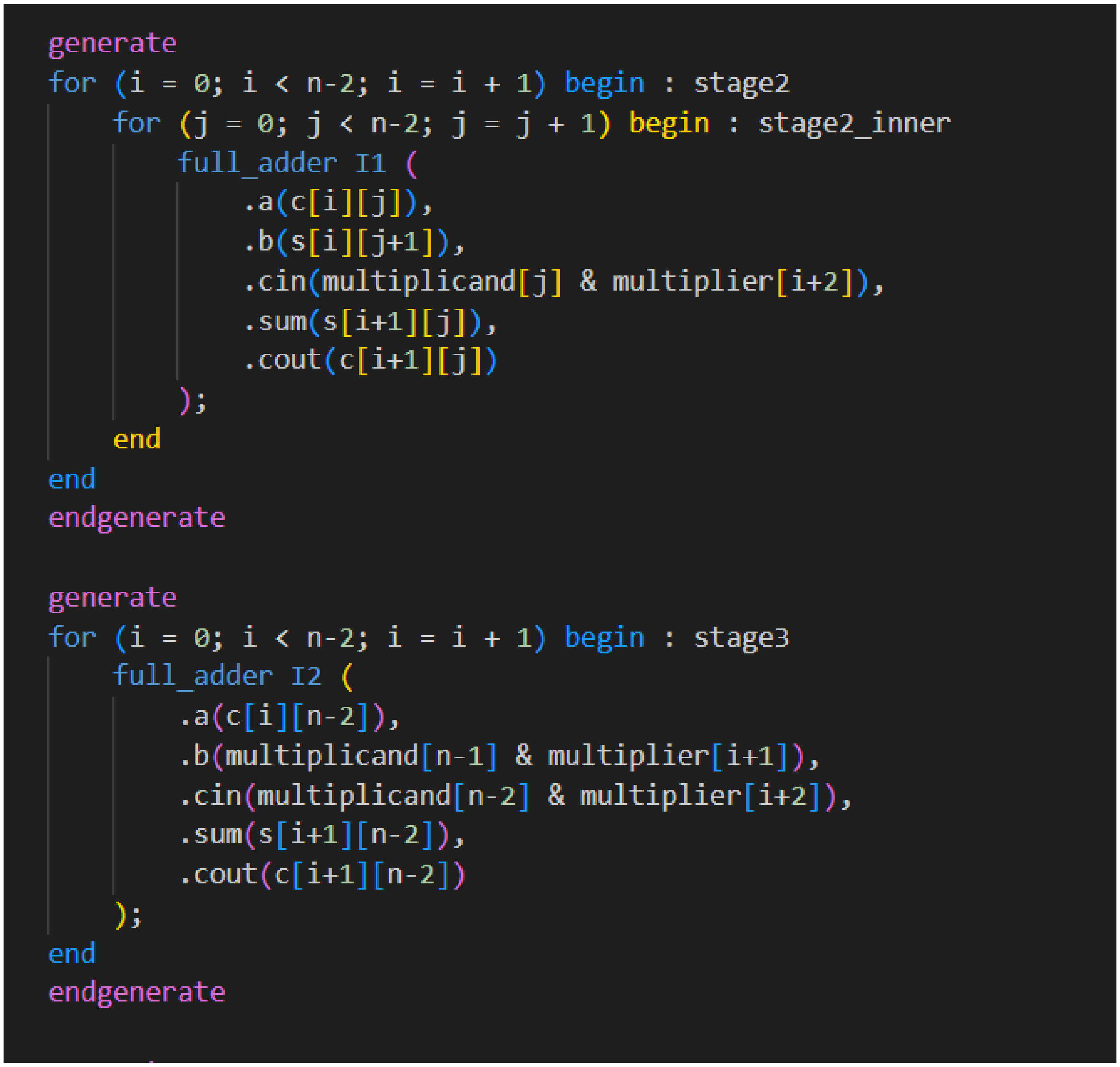
RTL:



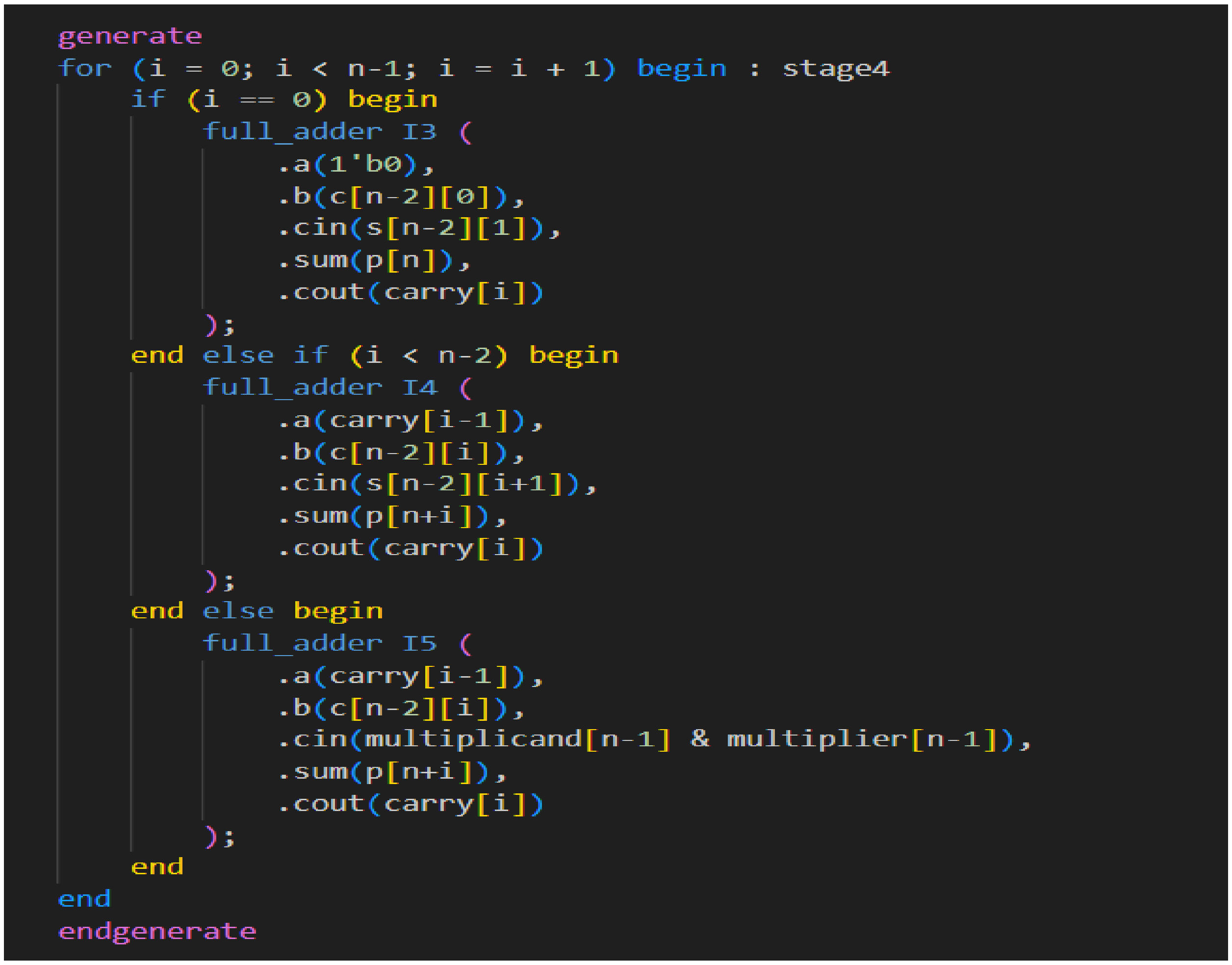
```
module full_adder (a,b,cin,sum,cout);
    input a,b,cin;
    output sum,cout;

    assign sum = a ^ b ^ cin;
    assign cout = (a & b) | (b & cin) | (a & cin);

endmodule
```



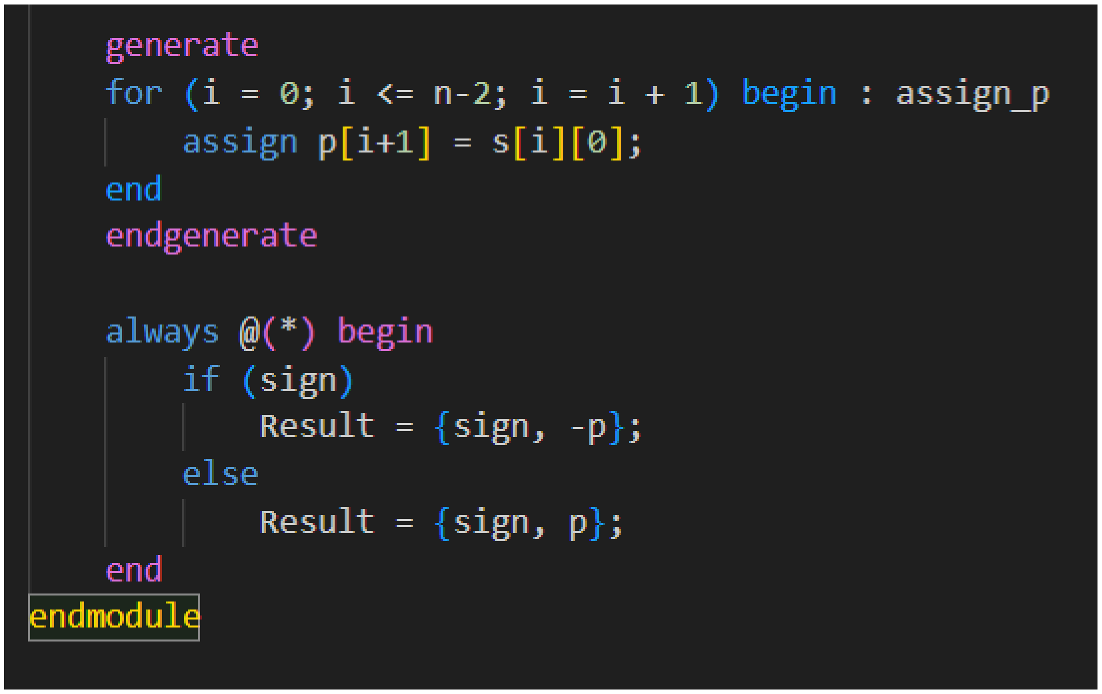
```
module Array_Multiplier (a, x, Result);
    parameter n = 5;
    input signed [n-1:0] a, x;
    output reg signed[2*n:0] Result;

    wire [n-1:0] multiplier, multiplicand;
    wire [n-2:0] c [0:n-2];
    wire [n-2:0] s [0:n-2];
    wire sign;
    wire [2*n-1:0] p;
    wire [n-2:0] carry;

    assign multiplier = x[n-1] ? -x : x;
    assign multiplicand = a[n-1] ? -a : a;
    assign p[0] = multiplicand[0] & multiplier[0];
    assign p[2*n-1] = carry[n-2];
    assign sign = x[n-1] ^ a[n-1];

    genvar i,j;
    generate
    for (i = 0; i < n-1; i = i + 1) begin : stage1
        full_adder I0 (
            .a(multiplicand[i+1] & multiplier[0]),
            .b(multiplicand[i] & multiplier[1]),
            .cin(1'b0),
            .sum(s[0][i]),
            .cout(c[0][i])
        );
    end
    endgenerate
```



```
generate
for (i = 0; i < n-2; i = i + 1) begin : stage2
    for (j = 0; j < n-2; j = j + 1) begin : stage2_inner
        full_adder I1 (
            .a(c[i][j]),
            .b(s[i][j+1]),
            .cin(multiplicand[j] & multiplier[i+2]),
            .sum(s[i+1][j]),
            .cout(c[i+1][j])
        );
    end
end
endgenerate

generate
for (i = 0; i < n-2; i = i + 1) begin : stage3
    full_adder I2 (
        .a(c[i][n-2]),
        .b(multiplicand[n-1] & multiplier[i+1]),
        .cin(multiplicand[n-2] & multiplier[i+2]),
        .sum(s[i+1][n-2]),
        .cout(c[i+1][n-2])
    );
end
endgenerate
```

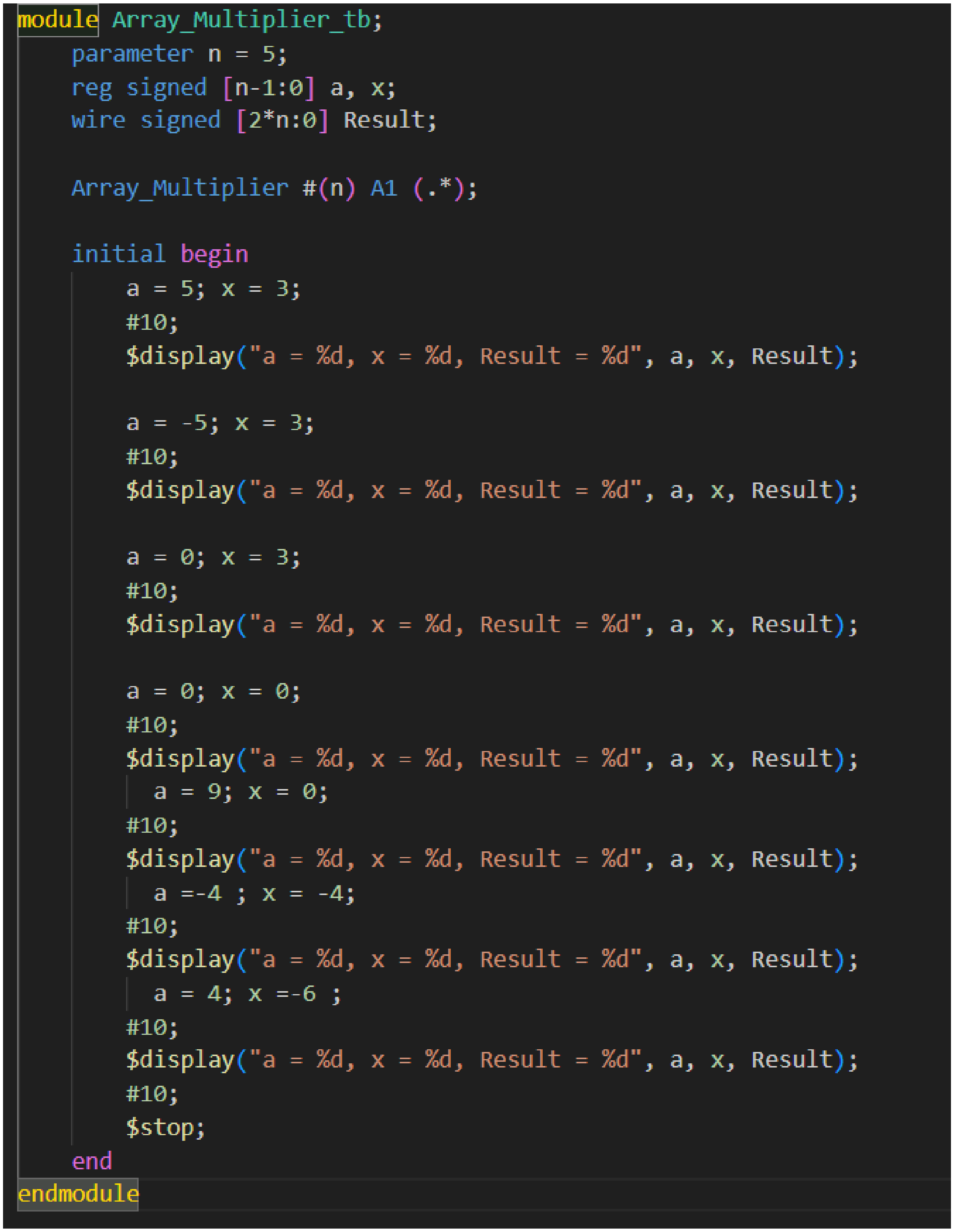


```
generate
for (i = 0; i < n-1; i = i + 1) begin : stage4
    if (i == 0) begin
        full_adder I3 (
            .a(1'b0),
            .b(c[n-2][0]),
            .cin(s[n-2][1]),
            .sum(p[n]),
            .cout(carry[i])
        );
    end else if (i < n-2) begin
        full_adder I4 (
            .a(carry[i-1]),
            .b(c[n-2][i]),
            .cin(s[n-2][i+1]),
            .sum(p[n+i]),
            .cout(carry[i])
        );
    end else begin
        full_adder I5 (
            .a(carry[i-1]),
            .b(c[n-2][i]),
            .cin(multiplicand[n-1] & multiplier[n-1]),
            .sum(p[n+i]),
            .cout(carry[i])
        );
    end
end
endgenerate
```



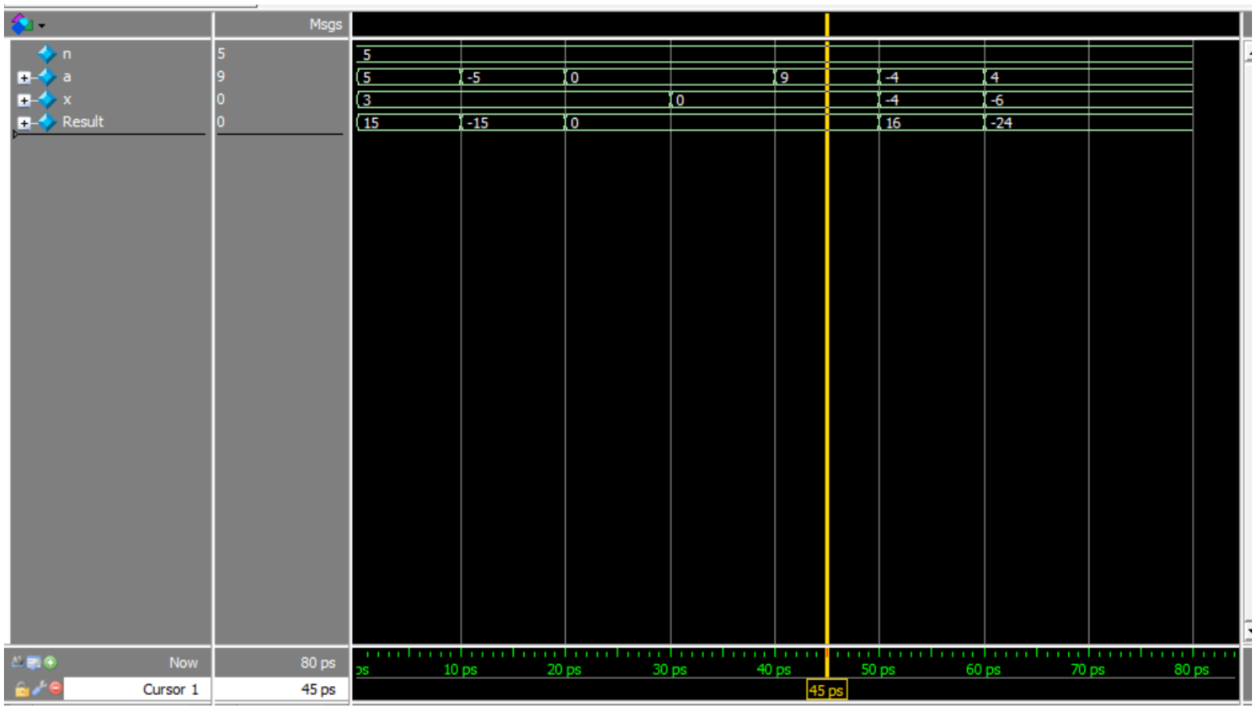
```
    generate
    for (i = 0; i <= n-2; i = i + 1) begin : assign_p
        assign p[i+1] = s[i][0];
    end
    endgenerate

    always @(*) begin
        if (sign)
            Result = {sign, -p};
        else
            Result = {sign, p};
    end
endmodule
```

Testbench:

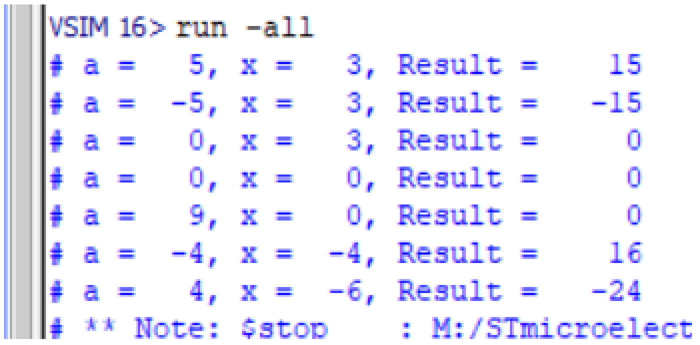


```
module Array_Multiplier_tb;
    parameter n = 5;
    reg signed [n-1:0] a, x;
    wire signed [2*n:0] Result;

    Array_Multiplier #(n) A1 (.*);

    initial begin
        a = 5; x = 3;
        #10;
        $display("a = %d, x = %d, Result = %d", a, x, Result);

        a = -5; x = 3;
        #10;
        $display("a = %d, x = %d, Result = %d", a, x, Result);

        a = 0; x = 3;
        #10;
        $display("a = %d, x = %d, Result = %d", a, x, Result);

        a = 0; x = 0;
        #10;
        $display("a = %d, x = %d, Result = %d", a, x, Result);
          a = 9; x = 0;
        #10;
        $display("a = %d, x = %d, Result = %d", a, x, Result);
          a =-4 ; x = -4;
        #10;
        $display("a = %d, x = %d, Result = %d", a, x, Result);
          a = 4; x =-6 ;
        #10;
        $display("a = %d, x = %d, Result = %d", a, x, Result);
        #10;
        $stop;
    end
endmodule
```

Wave:

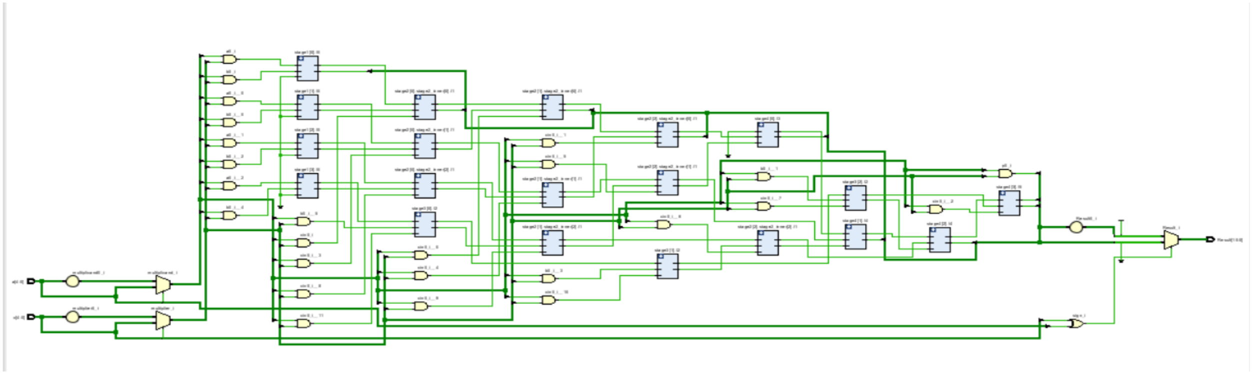


Transcript:

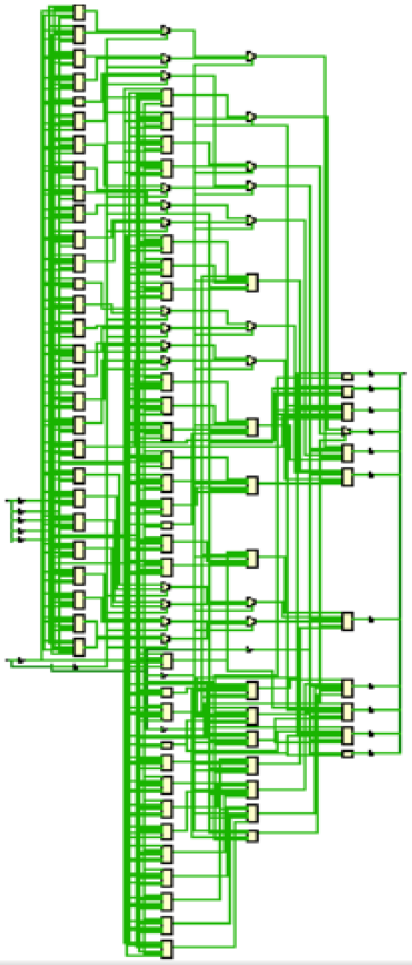


```
VSIM 16> run -all
# a =    5, x =    3, Result =    15
# a =   -5, x =    3, Result =   -15
# a =    0, x =    3, Result =     0
# a =    0, x =    0, Result =     0
# a =    9, x =    0, Result =     0
# a =   -4, x =   -4, Result =    16
# a =    4, x =   -6, Result =   -24
# ** Note: $stop    : M:/STmicroelect
```

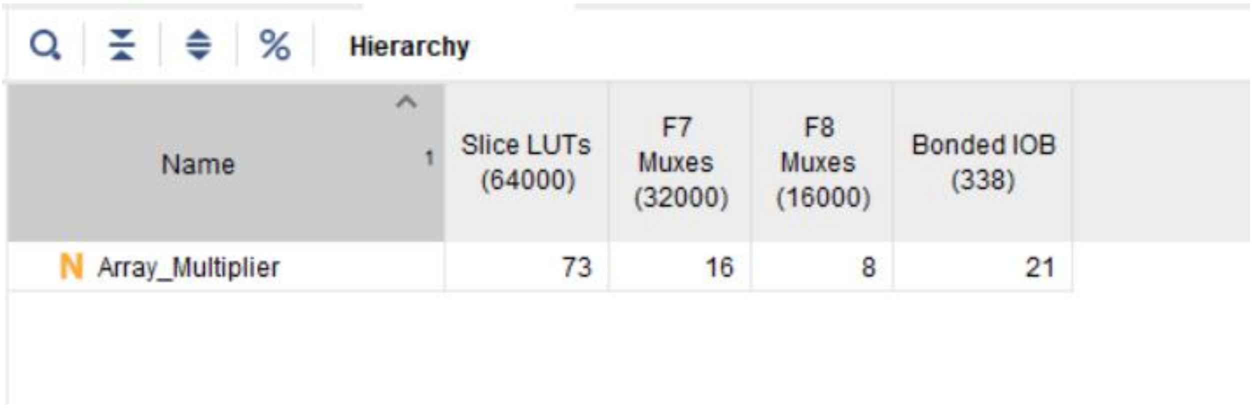
Schematic:



Synthesis:

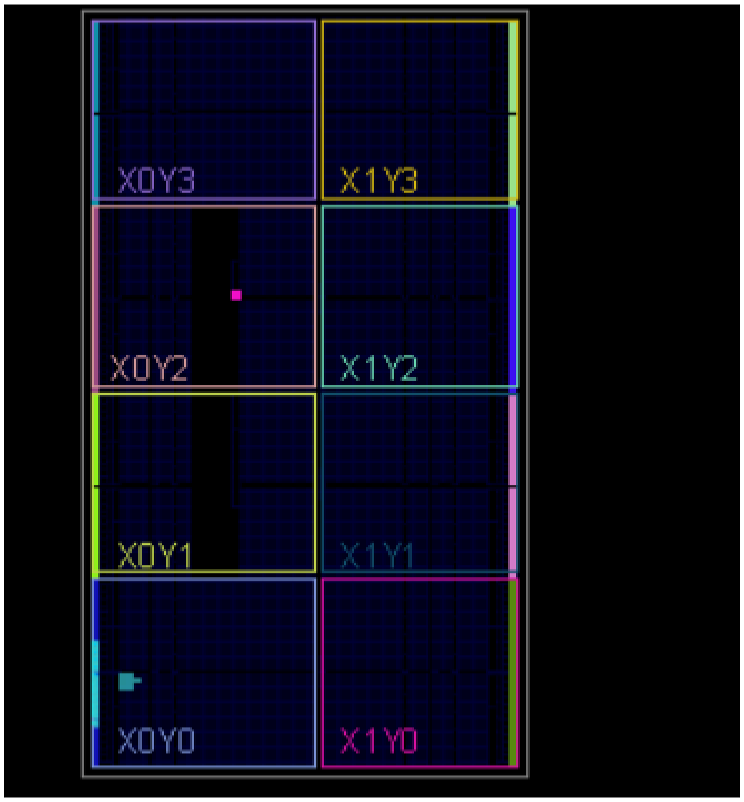


Utilization:



Name	Slice LUTs (64000)	F7 Muxes (32000)	F8 Muxes (16000)	Bonded IOB (338)
Array_Multiplier	73	16	8	21

Implementation:


X0Y3
X1Y3
X0Y2
X1Y2
X0Y1
X1Y1
X0Y0
X1Y0