যাদবপুর বিশ্ববিদ্যালয়
JADAVPUR UNIVERSITY

# SYNOPSYS LABORATORY ASSIGNMETS


**Name: Soham Pramanik**

**Roll No: 002110701150**

**Department: ETCE**

**Year: UG3**

## Task 1:

Implement signed 22x15 multiplier using 8x8 multiplier and adder logic. Write a Verilog testbench to verify the logical equivalence of 22x15 signed multiplier against reference RTL for 22x15 signed multiplier.

```
module mult2115(input signed [21:0] A,
                input signed [14:0] B,
                output signed [36:0] out);

        reg [7:0] a0, a1, a2;
        reg [7:0] b0, b1;

        reg [36:0] prod;
        reg [17:0] m;
        reg [36:0] temp1, temp2, temp3, temp4, temp5, temp6;

        always @(*) begin
                prod = 37'd0;

                a0 = {1'b0, A[6:0]};
                a1 = {1'b0, A[13:7]};
                a2 = A[21:14];

                b0 = {1'b0, B[6:0]};
                b1 = B[14:7];

                m = a0 * b0;
                temp1 = {20'd0, m};
                prod = prod + temp1;

                m = a1 * b0;
                temp2 = {13'd0, m, 7'd0};
                prod = prod + temp2;

                m = a2 * b0;
                temp3 = {6'd0, m, 14'd0};
                prod = prod + temp3;

                m = a0 * b1;
                temp4 = {13'd0, m, 7'd0};
                prod = prod + temp4;

                m = a1 * b1;
                temp5 = {6'd0, m, 14'd0};
                prod = prod + temp5;

                m = a2 * b1;
                temp6 = {3'd0, m, 21'd0};
                prod = prod + temp6;
        end

        assign out = prod;

endmodule
```

Alternate approach:

```
module mult2115(input signed [21:0] A,
                input signed [14:0] B,
                output signed [36:0] out);


        reg [7:0] a2, a1, a0;
        reg [7:0] b1, b0;

        reg [36:0] m1, m2, m3;

        always @ (*) begin
                a2 = A[21:14];
                a1 = {1'b0, A[13:7]};
                a0 = {1'b0, A[6:0]};

                b1 = B[14:7];
                b0 = {1'b0, B[6:0]};

                m1 = {3'b0, a2*b1, 21'b0} + {6'b0, a1*b1, 14'b0} + {13'b0,
a0*b1, 7'b0};
                m2 = {6'b0, a2*b0, 14'b0} + {13'b0, a1*b0, 7'b0} + {20'b0,
a0*b0};
                m3 = m1 + m2;
        end

        assign out = m3;
endmodule
```

**TEST BENCH**

```
`timescale 1ns / 100ps

module mult2115_tb;

        reg signed [21:0] A;
        reg signed [14:0] B;
        wire signed [36:0] out;

        mult2115 uut (
                .A(A),
                .B(B),
                .out(out)
        );

        initial begin
                $dumpfile("mult2115.vcd");
                $dumpvars(0, mult2115_tb);

                A = 'b1010101010101010101010;
                B = 'b10101010;
```

```
                // A = -9;
                // B = 8;

                #3;

                // Display results
                $display("A = %b, B = %b", A, B);
                $display("Out = %b", out);
                $display("Expected = %b", A * B);

                $finish;
        end

endmodule
```

```
A = 10101010101010101010, B = 00000010101010
Out = 00000000111000101010101010010011100100
Expected = 01010101010100111100100
$finish called from file "design.sv", line 32.
$finish at simulation time                   30
          V C S   S i m u l a t i o n   R e p o r t
Time: 3000 ps
CPU Time:      0.480 seconds;       Data structure size:   0.0Mb
```

Problems:

1. The above simulation is performed using the first code snippet. Until 22 bits, the output bitstream matches the expected bitstream, but beyond that it differs.

2. Using the second snippet results in incorrect output although functionally both snippets work on the same logic.

## Task 2:

Implement complex multiplication using 3 multipliers and 4 multipliers and compare logical equivalence using testbench simulation.

```
module complex3mul(input [3:0] A,
                   input [3:0] B,
                   input [3:0] C,
                   input [3:0] D,
                   output [7:0] Z_Re,
                   output [7:0] Z_Im);

        reg [7:0] temp1, temp2;
        reg [9:0] temp3;
        reg [4:0] a1, a2;
        reg [7:0] re, im;

        always @(*) begin

      temp1 = A * C;
      temp2 = B * D;
      a1 = A + B;
      a2 = C + D;
      temp3 = a1 * a2;

      re = temp1 - temp2;
      im = temp3 - temp1 - temp2;
        end

        assign Z_Re = re;
        assign Z_Im = im;

endmodule


module complex4mul (input [3:0] A,
                    input [3:0] B,
                    input [3:0] C,
                    input [3:0] D,
                    output [7:0] Z_Re,
                    output [7:0] Z_Im);

  reg [7:0] temp1, temp2, temp3, temp4, re, im;

  always @ (*) begin

    temp1 = A * C;
    temp2 = B * D;
    temp3 = A * D;
    temp4 = B * C;

    re = temp1 - temp2;
    im = temp3 + temp4;

  end
```

```
  assign Z_Re = re;
  assign Z_Im = im;

endmodule
```

**TEST BENCH**

```
`timescale 1ns/100ps

module complexMulTB;

        reg [3:0]Re1, Im1;
        reg [3:0]Re2, Im2;

        wire [7:0] Mul3r, Mul3i;
        wire [7:0] Mul4r, Mul4i;

        complex3mul uut1(Re1, Im1, Re2, Im2, Mul3r, Mul3i);
        complex4mul uut2(Re1, Im1, Re2, Im2, Mul4r, Mul4i);

        initial begin
        $dumpfile("complexMul.vcd");
        $dumpvars(0, complexMulTB);

        Re1 = 4'b1101;
        Im1 = 4'b1010;
        Re2 = 4'b0011;
        Im2 = 4'b0110;

        #20
        $display("");
                $display("Complex Number 1 = %d + j%d", Re1, Im1);
        $display("Complex Number 2 = %d + j%d", Re2, Im2);

        #20 $display("Product_3 = %d + j%d", Mul3r, Mul3i);
        #20 $display("Product_4 = %d + j%d", Mul4r, Mul4i);
        $display("");
        end

endmodule
```

```
Complex Number 1 = 13 + j10
Complex Number 2 =  3 + j 6
Product_3 = 235 + j108
Product_4 = 235 + j108

          V C S   S i m u l a t i o n   R e p o r t
Time: 60000 ps
CPU Time:      0.540 seconds;       Data structure size:   0.0Mb
```

## Task 4:

Write RTL for the following logic and compare post synthesis timing.

a) (flops)-> (operation a)-> (operation b)-> (flops)-> (flops)
b) (flops) -> (operation a)->(flops)->(operation b)->(flops)

Assume that flops have no initial value applied by the user.

operation_a is 8 bits + 8 bits addition; operation _b is reduction ^ previous stages (output of the adder).

```
module function_a(input [7:0] data_in,
                  input clk,
                  input rst,
                  output data_out);


        reg [7:0] flop1;
    reg flop2, flop3;

    wire [7:0] add;
    wire reduc;

    assign add = flop1 + 8'h55;
    assign x_pd = add[0] ^ add[1] ^ add[2] ^ add[3] ^ add[4] ^ add[5] ^
add[6] ^ add[7];

    always @(posedge clk) begin

      if (rst) begin
        flop1 <= 8'd0;
        flop2 <= 1'b0;
        flop3 <= 1'b0;
      end

      else begin
        flop1 <= data_in;
        flop2 <= reduc;
        flop3 <= flop2;
      end

    end

    assign data_out = flop3;
endmodule

module function_b(input [7:0] data_in,
                  input clk,
                  input rst,
                  output data_out);

        reg [7:0] flop1, flop2;
        reg flop3;

        wire [7:0] add;
        wire reduc;
```

```
    assign add = flop1 + 8'h55;
    assign reduc = flop2[0] ^ flop2[1] ^ flop2[2] ^ flop2[3] ^ flop2[4] ^
flop2[5] ^ flop2[6] ^ flop2[7];

    always @(posedge clk) begin

      if (rst) begin
        flop1 <= 8'd0;
        flop2 <= 8'd0;
        flop3 <= 1'b0;
      end

      else begin
        flop1 <= data_in;
        flop2 <= add;
        flop3 <= reduc;
      end

    end

    assign data_out = flop3;
endmodule
```

**TEST BENCH**

```
`timescale 1ns/100ps

module function_tb;

        reg clk,rst;
        reg [7:0] data_in;

        wire data_out1;
        wire data_out2;

        //function_a uut1(data_in, clk, rst, data_out1);
        function_b uut2(data_in, clk, rst, data_out2);

        always #5 clk = ~clk;

    initial begin

        $dumpfile("waveform.vcd");
        $dumpvars(0, function_tb);

        clk = 1'b0;
        rst = 1'b1;

        #5 rst = 1'b0;

        data_in = 8'b11001010;
        #10 rst = 1'b1;

                data_in = 8'b01010110;
```

```
        #5 rst = 1'b0;

        $finish;
    end
endmodule
```

Note: For some reason the timing analysis reports were identical for both function_a and function_b last time I checked. Need to re-check when I get back to lab.