

**CS331**

**Computers Organization & Architecture**

**Assignment\_2**

**Problem Solving**

**FP Multiplier**

**Coding:**

module try\_mult (floatp\_A, floatp\_B, sign, exponent,

exp\_unbiased, exp\_sum, prod,sum);

input [31:0] floatp\_A, floatp\_B;

output sign;

output [7:0] exponent, exp\_unbiased;

output [8:0] exp\_sum;

output [22:0] prod;

output [31:0] sum ;

reg sign\_a, sign\_b;

reg [7:0] exp\_a, exp\_b;

reg [7:0] exp\_a\_bias, exp\_b\_bias;

reg [8:0] exp\_sum;

reg [22:0] fract\_a, fract\_b;

reg [45:0] prod\_dbl;

reg [22:0] prod;

reg sign;

reg [31:0] sum ;

reg [7:0] exponent, exp\_unbiased;

//define sign, exponent, and fraction

always @ (floatp\_A or floatp\_B)

begin

sign\_a = floatp\_A[31];

sign\_b = floatp\_B[31];

exp\_a = floatp\_A[30:23];

exp\_b = floatp\_B[30:23];

fract\_a = floatp\_A[22:0];

fract\_b = floatp\_B[22:0];

//bias exponents

exp\_a\_bias = exp\_a + 8'b0111\_1111;

exp\_b\_bias = exp\_b + 8'b0111\_1111;

//add exponents

exp\_sum = exp\_a\_bias + exp\_b\_bias;

//remove one bias

exponent = exp\_sum - 8'b0111\_1111;

exp\_unbiased = exponent - 8'b0111\_1111;

//multiply fractions

//if (flp\_a != 0 || flp\_b!=0) begin

prod\_dbl = fract\_a \* fract\_b;

prod = prod\_dbl[45:23];

//postnormalize product

/\*for (int i = 0; i < 22;i = i + 1) // This will not synthesize!

begin

prod = prod << 1;

exp\_unbiased = exp\_unbiased - 1;

end\*/

sign = sign\_a ^ sign\_b;

if (prod ==0) begin

sum =32'b0;

end else sum ={sign, exp\_unbiased, prod};

end

//end

endmodule

module try\_mult\_test;

reg [31:0]floatp\_A,floatp\_B;

reg sign;

wire [7:0] exponent, exp\_unbiased;

wire [8:0] exp\_sum;

wire [22:0] prod;

wire [31:0] sum ;

initial

begin

$monitor("in1 = %b , in2 = %b ,out1 = %b ,out2 = %b ,out3 = %b ,out4 = %b ,out5 = %b" ,floatp\_A, floatp\_B, sign, exponent,

exp\_unbiased, exp\_sum, prod,sum);

#10

floatp\_A[31:0]=32'b0\_0000\_0011\_1010\_0000\_0000\_0000\_0000\_000;

;

floatp\_B[31:0]=32'b0\_0000\_0010\_1100\_0000\_0000\_0000\_0000\_000;

#10

floatp\_A[31:0]=1;

floatp\_B[31:0]=1;

#10

floatp\_A[31:0]=1;

floatp\_B[31:0]=0;

#10

floatp\_A[31:0]=0;

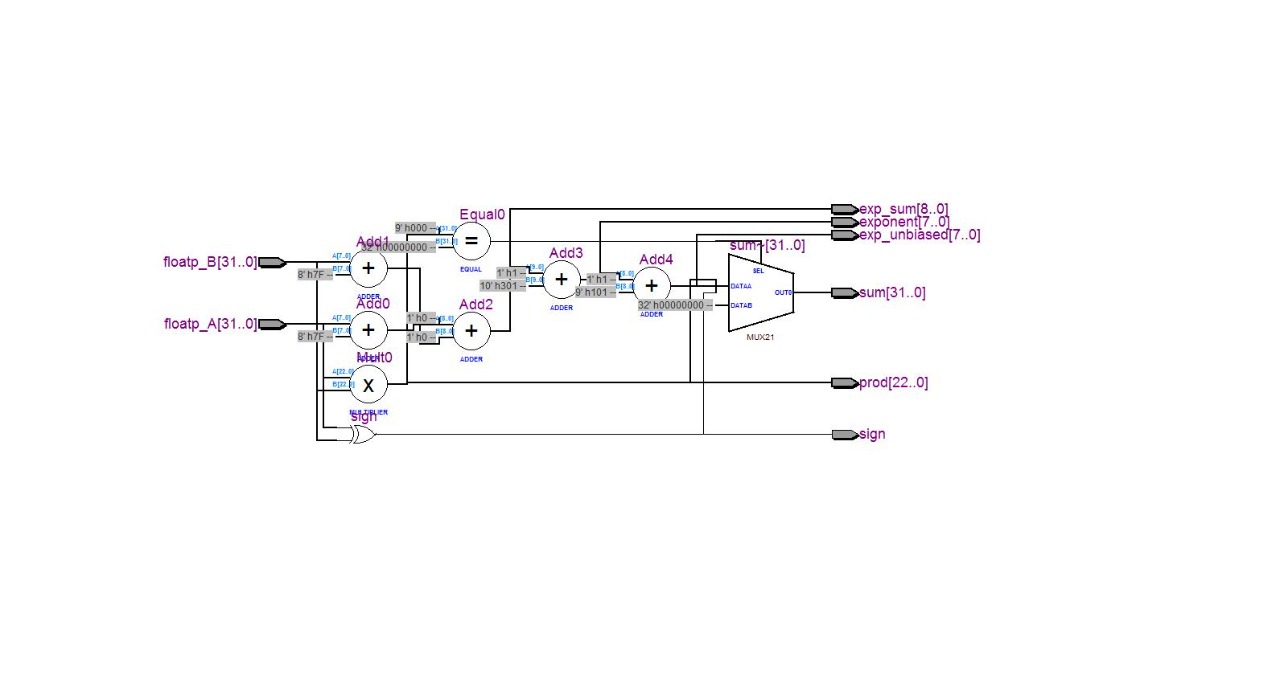
floatp\_B[31:0]=0;

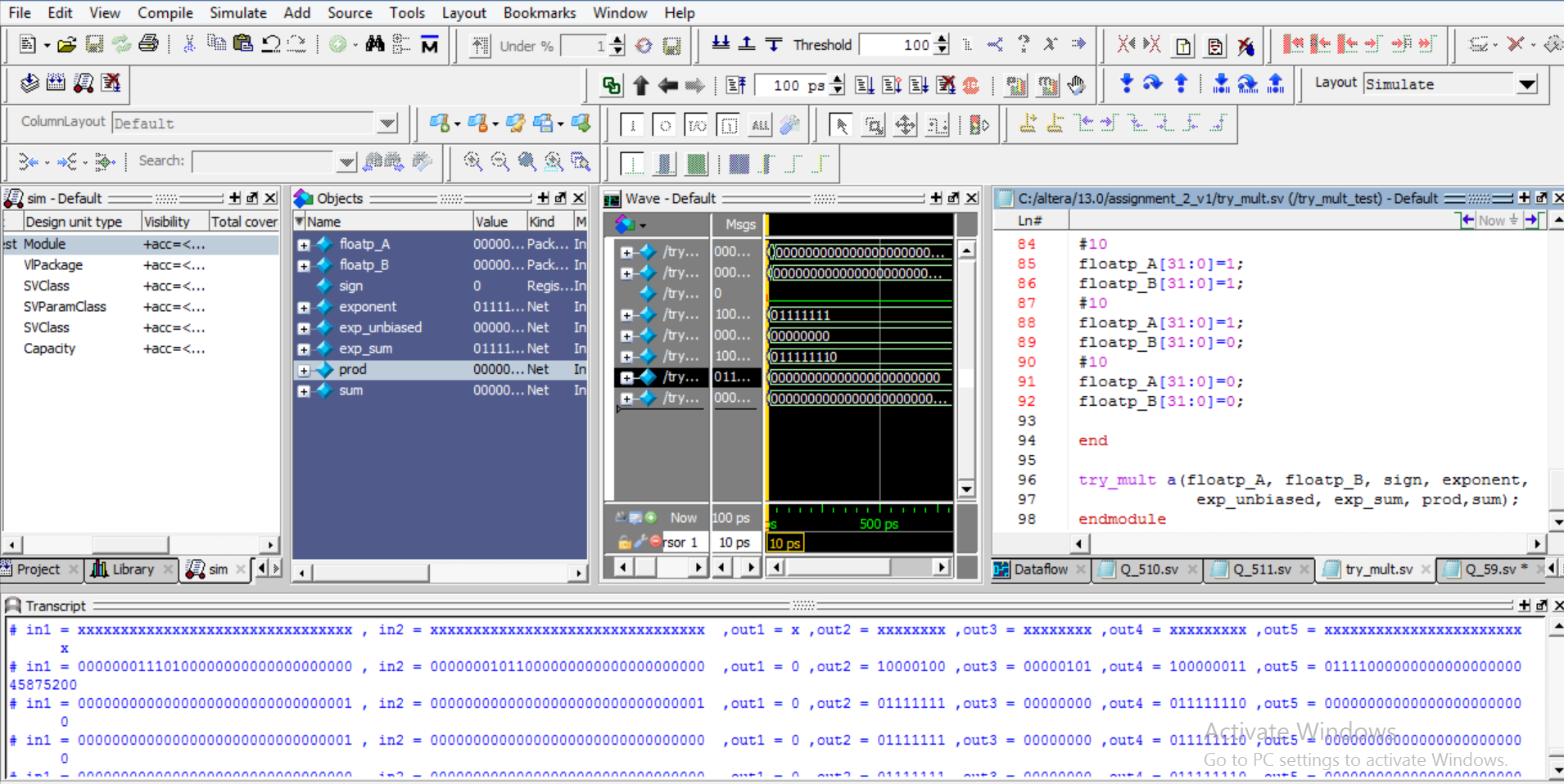
end

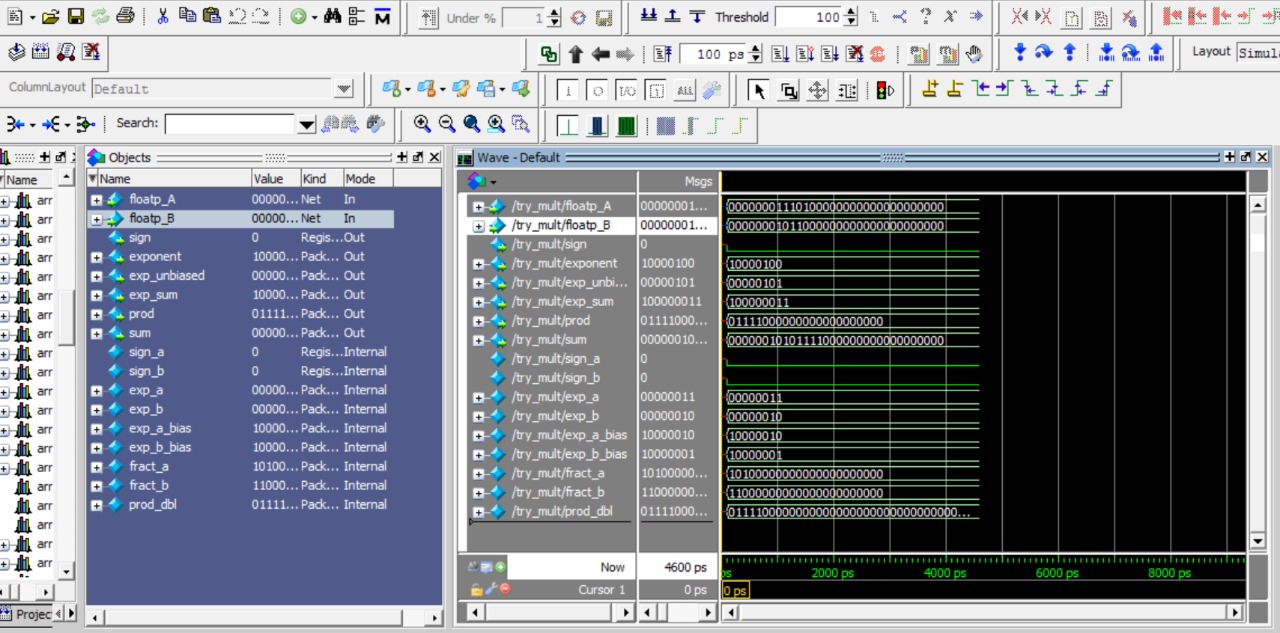
try\_mult a(floatp\_A, floatp\_B, sign, exponent,

exp\_unbiased, exp\_sum, prod,sum);

endmodule



****

****

**ALU**

**Exercise 5.9:**

**Coding:**

module Q\_59(input logic [31:0] A, B,

input logic [2:0] F,

output logic [31:0] Y);

logic [31:0] S, Bout;

assign Bout = F[2] ? ~B : B;

assign S = A + Bout + F[2];

always\_comb

case (F[1:0])

2'b00: Y <= A & Bout;

2'b01: Y <= A | Bout;

2'b10: Y <= S;

2'b11: Y <= S[31];

endcase

endmodule

module TestModule;

reg [31:0]A,B;

reg [2:0]F;

wire [31:0]Y;

initial

begin

$monitor("in1 = %b , in2 = %b ,in3 = %b ,out1 = %b" ,A,B,F,Y);

#10

A[31:0]=0;

B[31:0]=1;

F[2:0]=0;

#10

A[31:0]=0;

B[31:0]=0;

F[2:0]=0;

#10

A[31:0]=1;

B[31:0]=1;

F[2:0]=0;

#10

A[31:0]=1;

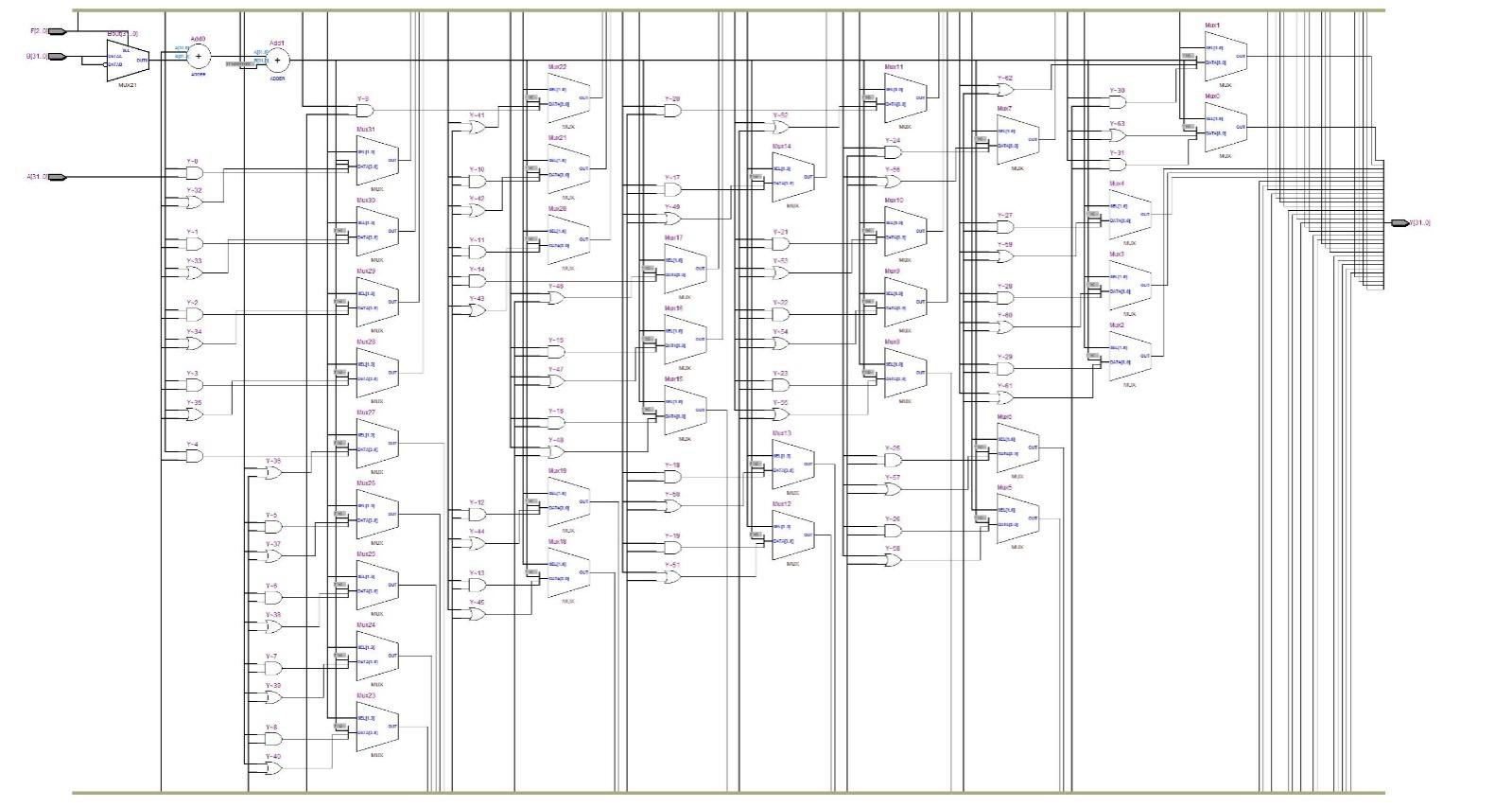
B[31:0]=1;

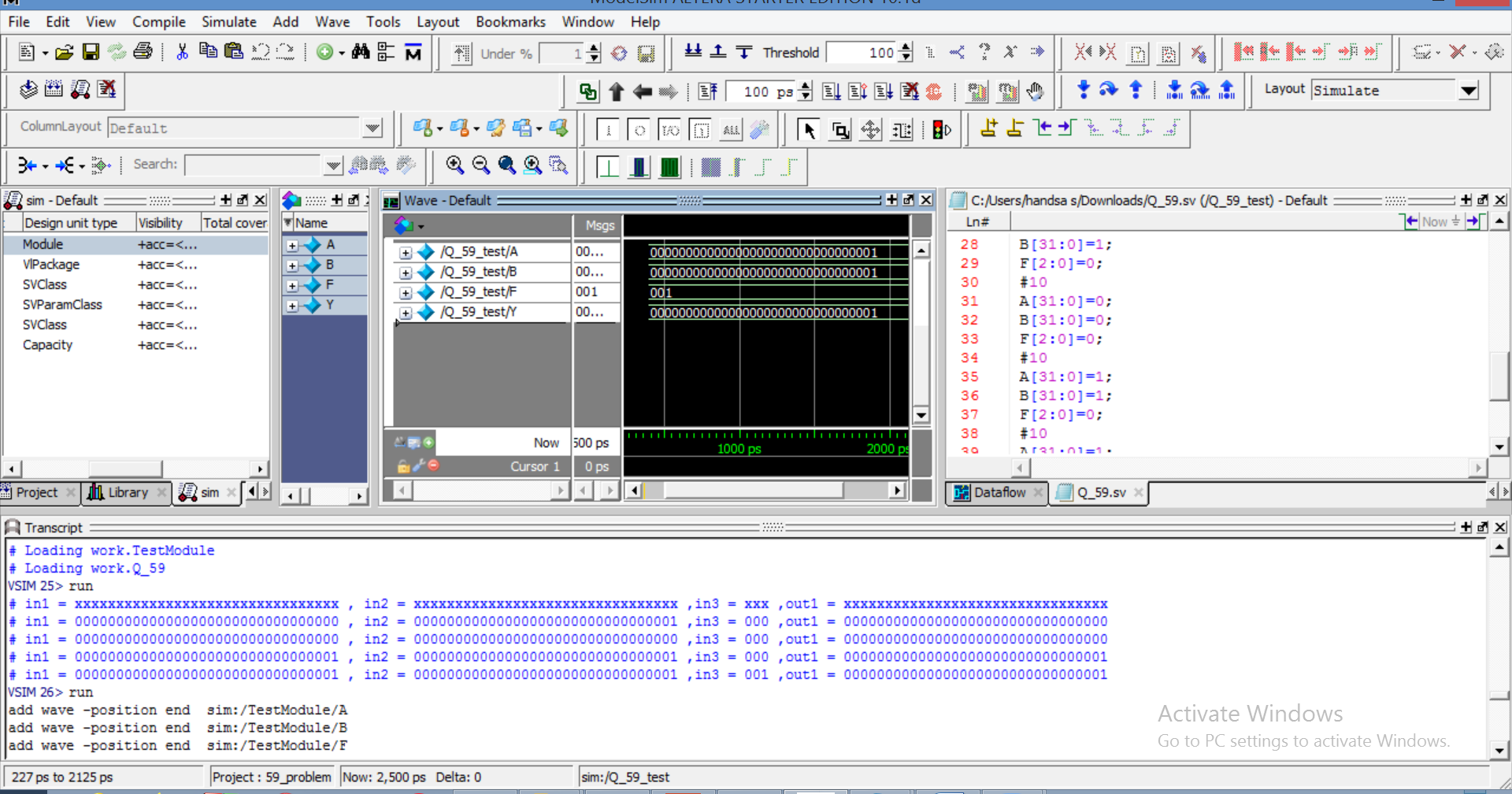
F[2:0]=1;

end

Q\_59 a(A,B,F,Y);

endmodule



****

**Exercise 5.10:**

**Coding:**

module Q\_510(input logic [31:0] A, B,

input logic [2:0] F,

output logic [31:0] Y,

output logic Overflow);

logic [31:0] S, Bout;

assign Bout = F[2] ? ~B : B;

assign S = A + Bout + F[2];

always\_comb

case (F[1:0])

2'b00: Y = A & Bout;

2'b01: Y = A | Bout;

2'b10: Y = S;

2'b11: Y = S[31];

endcase

always\_comb

case (F[2])

1'b0: Overflow = A[31] & B[31] & ~S[31] |

~A[31] & ~B[31] & S[31];

1'b1: Overflow = ~A[31] & B[31] & S[31] |

A[31] & ~B[31] & ~S[31];

default: Overflow = 1'b0;

endcase

endmodule

module Q\_510\_test;

reg [31:0]A,B;

reg [2:0]F;

wire [31:0]Y;

wire Overflow;

initial

begin

$monitor("in1 = %b , in2 = %b ,in3 = %b ,out1 = %b ,out2 = %b" ,A,B,F,Y,Overflow);

#10

A[31:0]=0;

B[31:0]=1;

F[2:0]=0;

#10

A[31:0]=0;

B[31:0]=0;

F[2:0]=0;

#10

A[31:0]=1;

B[31:0]=1;

F[2:0]=0;

#10

A[31:0]=1;

B[31:0]=1;

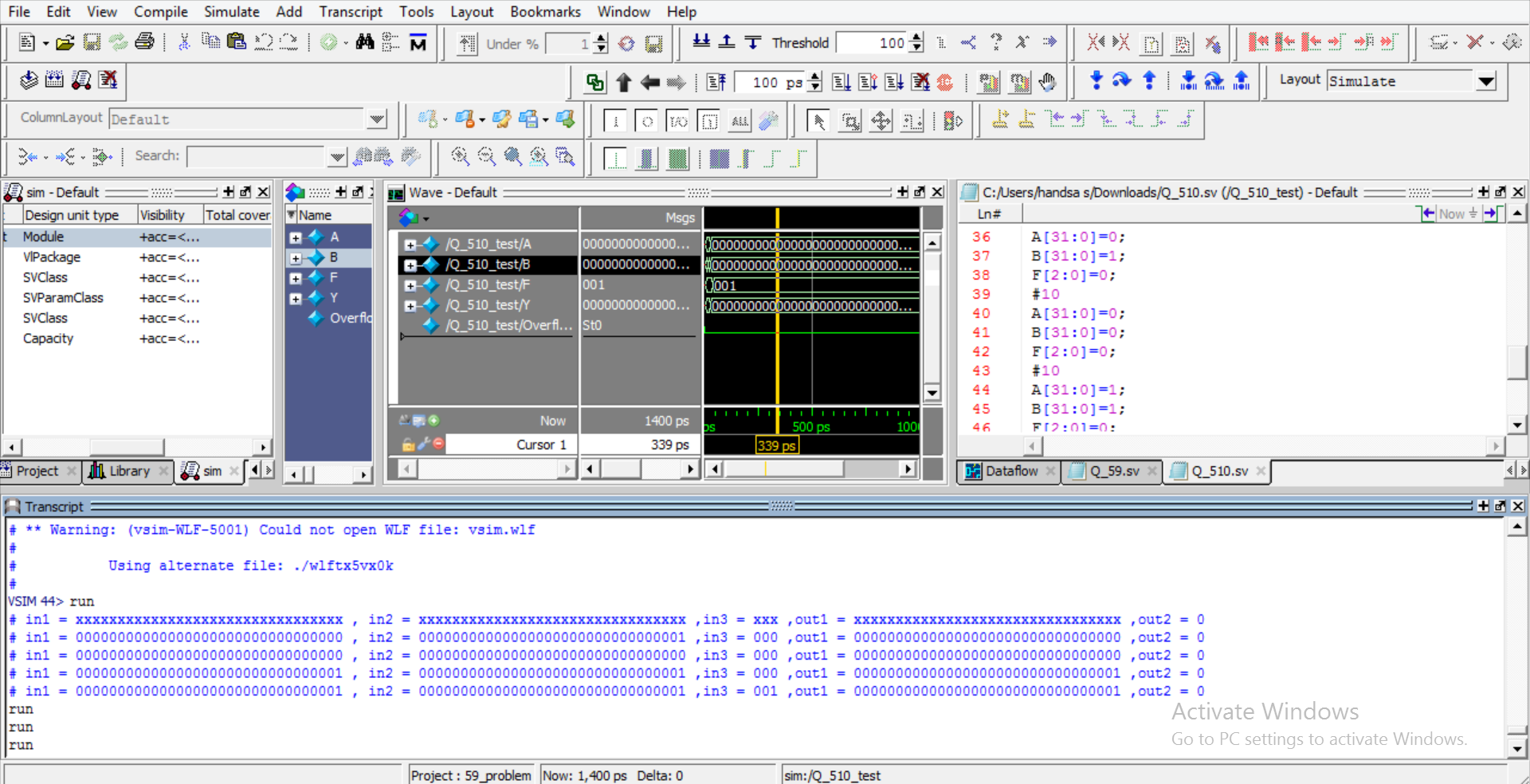
F[2:0]=1;

end

Q\_510 a(A,B,F,Y,Overflow);

endmodule





**Exercise 5.11:**

**Coding:**

module Q\_511(input logic [31:0] A, B,

input logic [2:0] F,

output logic [31:0] Y,

output logic Zero, Overflow);

logic [31:0] S, Bout;

assign Bout = F[2] ? ~B : B;

assign S = A + Bout + F[2];

always\_comb

case (F[1:0])

2'b00: Y <= A & Bout;

2'b01: Y <= A | Bout;

2'b10: Y <= S;

2'b11: Y <= S[31];

endcase

assign Zero = (Y == 32'b0);

always\_comb

case (F[2:1])

2'b01: Overflow <= A[31] & B[31] & ~S[31] |

~A[31] & ~B[31] & S[31];

2'b11: Overflow <= ~A[31] & B[31] & S[31] |

A[31] & ~B[31] & ~S[31];

default: Overflow <= 1'b0;

endcase

endmodule

module Q\_511\_test;

reg [31:0]A,B;

reg [2:0]F;

wire [31:0]Y;

wire Overflow,Zero;

initial

begin

$monitor("in1 = %b , in2 = %b ,in3 = %b ,out1 = %b ,out2 = %b" ,A,B,F,Y,Overflow,Zero);

#10

A[31:0]=0;

B[31:0]=1;

F[2:0]=0;

#10

A[31:0]=0;

B[31:0]=0;

F[2:0]=0;

#10

A[31:0]=1;

B[31:0]=1;

F[2:0]=0;

#10

A[31:0]=1;

B[31:0]=1;

F[2:0]=1;

end

Q\_511 a(A,B,F,Y,Overflow,Zero);

endmodule



