Virginia Tech Training

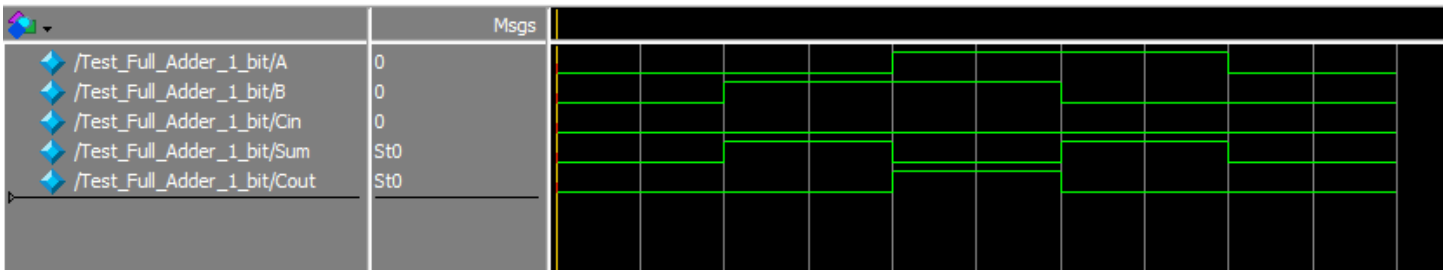
Name: Kholoud Ebrahim Ali Darwesh

1. Write a behavioral model of a single bit full adder.

```
1  module Full_Adder_1_bit (A, B, Cin, Sum, Cout);
2  input A, B, Cin;
3  output reg Sum, Cout;
4  // behavioral model of a single bit full adder
5  always @(*) begin
6            Sum  = A ^ B ^ Cin;
7            Cout =(A & B)|(B & Cin)|(A & Cin);
8  end
9  endmodule
```

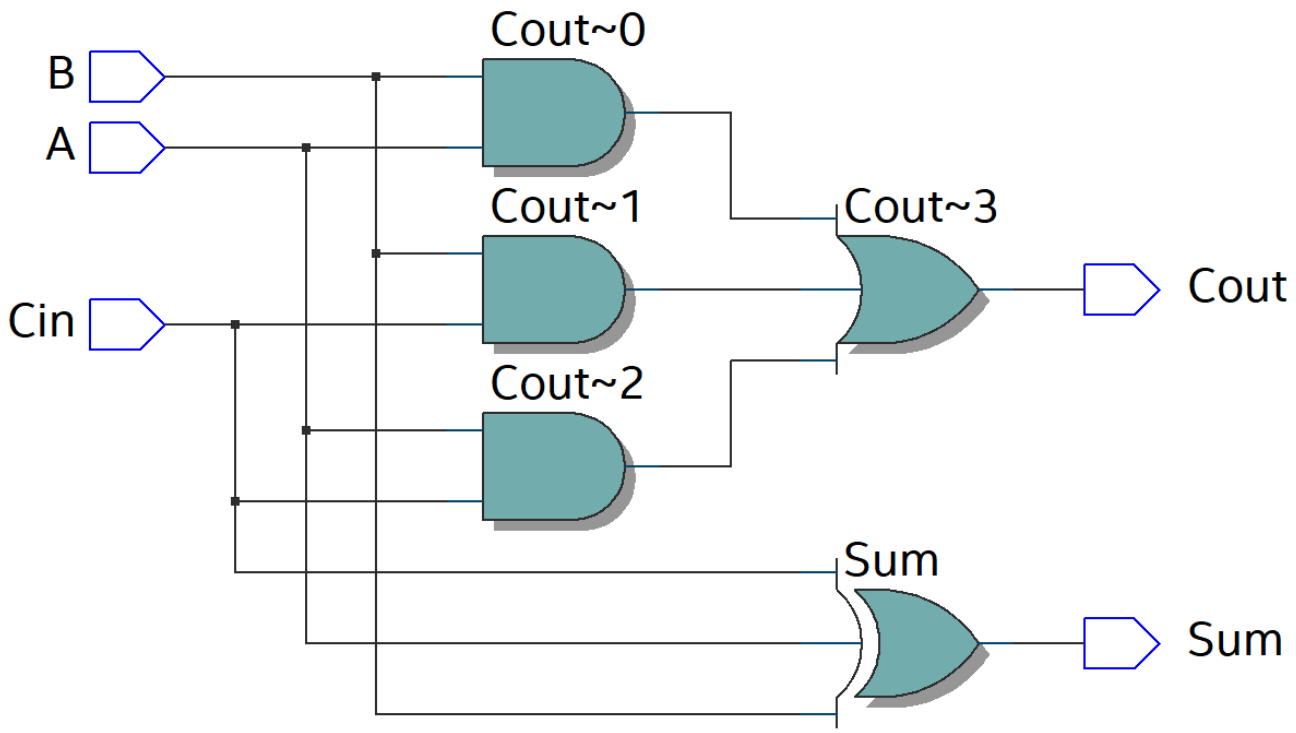
```
1  `timescale 100ps/1ps
2  module Test_Full_Adder_1_bit ;
3  reg  A, B, Cin;
4  wire Sum, Cout;
5  // Testing of behavioral model of a single bit full adder
6  Full_Adder_1_bit DUT1(A, B, Cin, Sum, Cout);
7
8  initial begin
9      Cin=1'b0;
10     A = 1'b0;
11     B = 1'b0;
12   #1
13     B = 1'b1;
14   #1
15     A = 1'b1;
16   #1
17     B = 1'b0;
18   #1
19     A = 1'b0;
20 // it takes only 500ps in simulation
21 end
22
23 initial begin
24     $monitor ("time:%2d   A: %b   B:%b   Sum: %b   Cout: %b\n",
25               $time     , A      , B     , Sum      , Cout        );
26 end
27 endmodule
```

The simulation of the first module




```
# time: 0   A: 0   B:0   Sum: 0   Cout: 0
#
# time: 1   A: 0   B:1   Sum: 1   Cout: 0
#
# time: 2   A: 1   B:1   Sum: 0   Cout: 1
#
# time: 3   A: 1   B:0   Sum: 1   Cout: 0
#
# time: 4   A: 0   B:0   Sum: 0   Cout: 0
#
```

The RTL of the first module

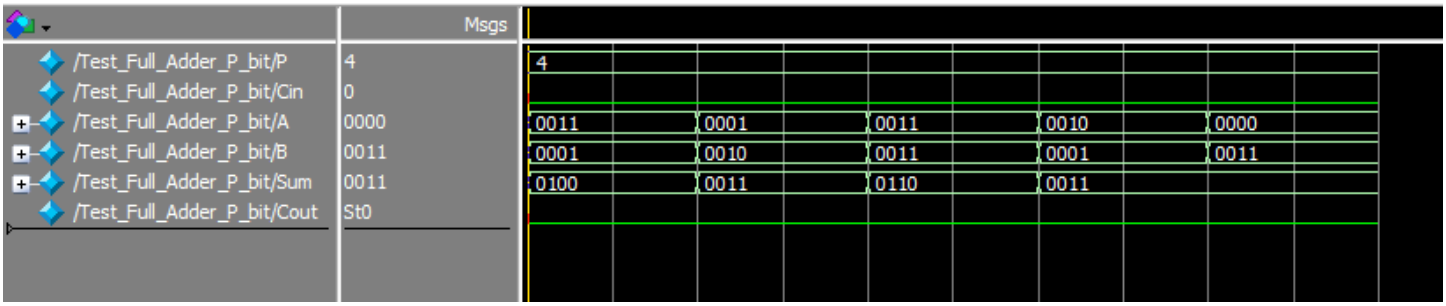


2. Write a structural model of P-bit full adder.

```
module Full_Adder_P_bit #(parameter P=4)(A, B, Cin, Sum, Cout);
input  Cin;
input  [P-1:0]A, B;
output [P-1:0]Sum;
output Cout;
wire   [P:0]C_internal;
// Structural model of P-bit full adder
genvar i;
generate
for (i=0; i< P ; i = i+1) begin : generate_P_bit_FA
    Full_Adder_1_bit DUT0(.A(A[i]), .B(B[i]), .Cin(C_internal[i]),
                          .Sum(Sum[i]), .Cout(C_internal[i+1]));
end
endgenerate

assign C_internal[0] = Cin;
assign Cout = C_internal[P];
endmodule
```

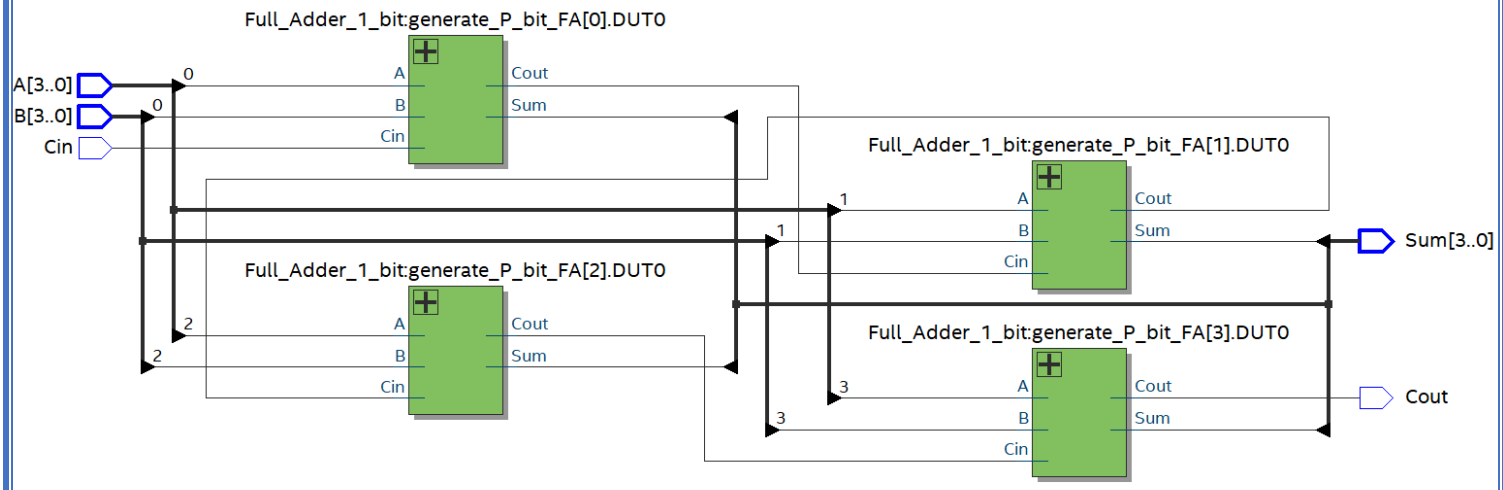
```
`timescale 100ps/1ps
module Test_Full_Adder_P_bit;
reg Cin;
parameter P = 4;
reg  [P-1:0]A, B;
wire [P-1:0]Sum;
wire Cout;
// Testing of Structural model of P-bit full adder
Full_Adder_P_bit #(.P(P)) DUT(A, B, Cin, Sum, Cout);

initial begin
    Cin= 0;
    A= 3; B= 1;
    #1
    A= 1; B= 2;
    #1
    A= 3; B= 3;
    #1
    A= 2; B= 1;
    #1
    A= 0; B= 3;
end

initial begin
$monitor ("time: %2d   A: %b   B: %b   Sum: %b   Cout: %b",
          $time       , A     ,  B     ,  Sum     ,  Cout      );
end
endmodule
```

The simulation of the second module




```
VSIM 8> run
# time:  0   A: 0011   B: 0001   Sum: 0100   Cout: 0
# time:  1   A: 0001   B: 0010   Sum: 0011   Cout: 0
# time:  2   A: 0011   B: 0011   Sum: 0110   Cout: 0
# time:  3   A: 0010   B: 0001   Sum: 0011   Cout: 0
# time:  4   A: 0000   B: 0011   Sum: 0011   Cout: 0
```

The RTL of the second module

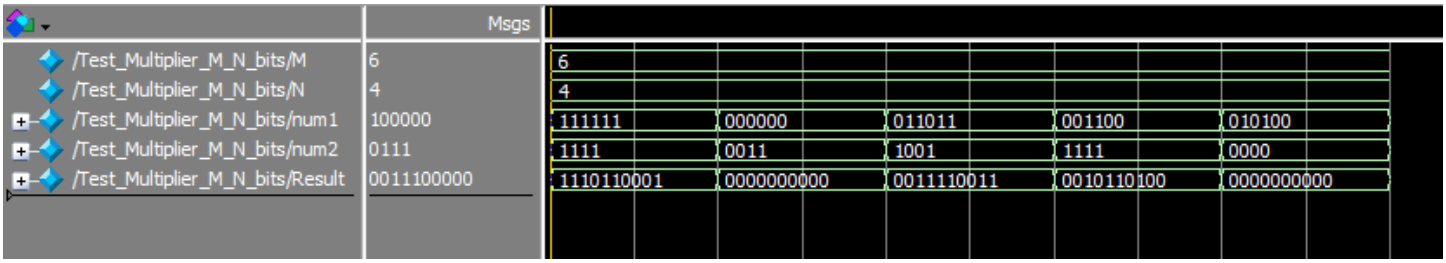


3. Write a behavioral model of M-bit by N-bit Multiplier.

```
module Multiplier_M_N_bits #(parameter M=4, N=3)(num1, num2, Result);
input  [M-1:0]num1;
input  [N-1:0]num2;
output reg [M+N-1:0]Result;
reg    [M+N-1:0]Mul_stage;
reg    [M+N-1:0]Mul_Register[N-1:0];
integer i;
// behavioral model of M-bit by N-bit Multiplier
always @(*) begin
Result =0;
    for (i=0; i< N ; i = i+1) begin :Multiplier_Register
        Mul_stage = num1 & {M{num2[i]}};
        Mul_stage = Mul_stage << i;
        Mul_Register[i]= Mul_stage;
        Result= Mul_Register[i]+Result;
    end
end
endmodule
```

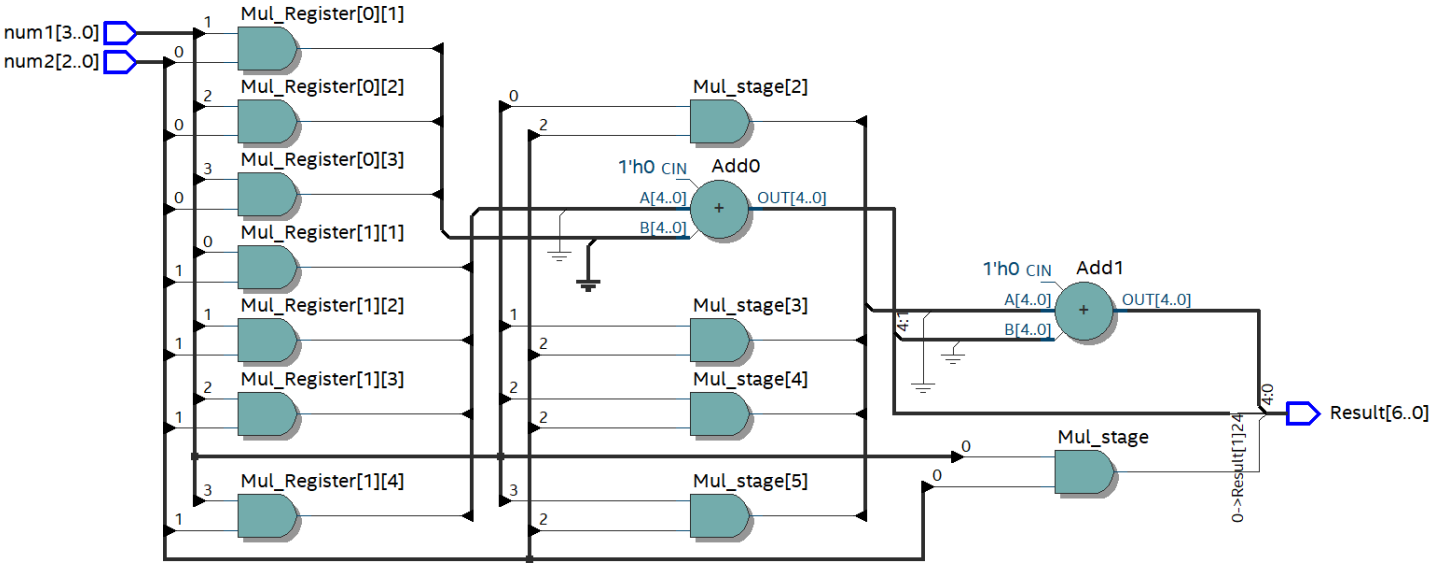
```
`timescale 100ps/1ps
module Test_Multiplier_M_N_bits;
parameter M= 6,N= 4;
reg  [M-1:0]num1;
reg  [N-1:0]num2;
wire [M+N-1:0]Result;

Multiplier_M_N_bits #(.M(M), .N(N)) DUT(num1, num2, Result);

initial begin
    num1 = 63; num2 = 15;
    #1
    num1 =  0; num2 =  3;
    #1
    num1 = 27; num2 =  9;
    #1
    num1 = 12; num2 = 31;
    #1
    num1 = 20; num2 = 0;
    #1
    num1 = 32; num2 = 7;
end

initial begin
    $monitor("time: %2d   num1: (%d)%b   num2: (%d)%b   Result: (%d)%b",
             $time        ,num1, num1    ,num2, num2    ,Result, Result );
end
endmodule
```

The simulation of the third module

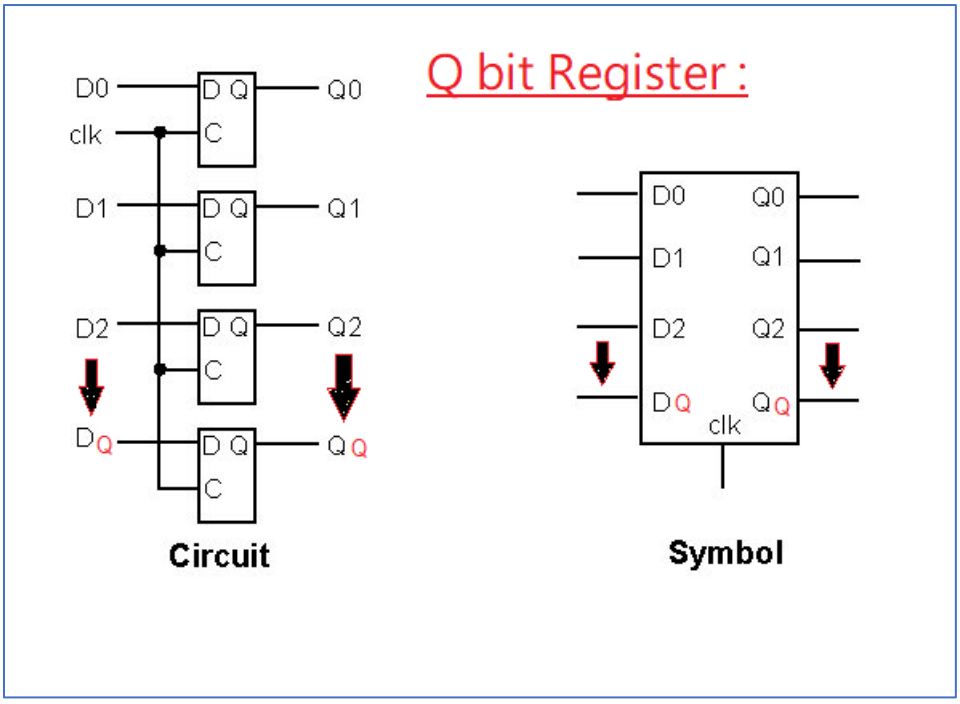



```
# time:  0   num1: (63)111111   num2: (15)1111   Result: ( 945)1110110001
# time:  1   num1: ( 0)000000   num2: ( 3)0011   Result: (   0)0000000000
# time:  2   num1: (27)011011   num2: ( 9)1001   Result: ( 243)0011110011
# time:  3   num1: (12)001100   num2: (15)1111   Result: ( 180)0010110100
# time:  4   num1: (20)010100   num2: ( 0)0000   Result: (   0)0000000000
```

The RTL of the third module



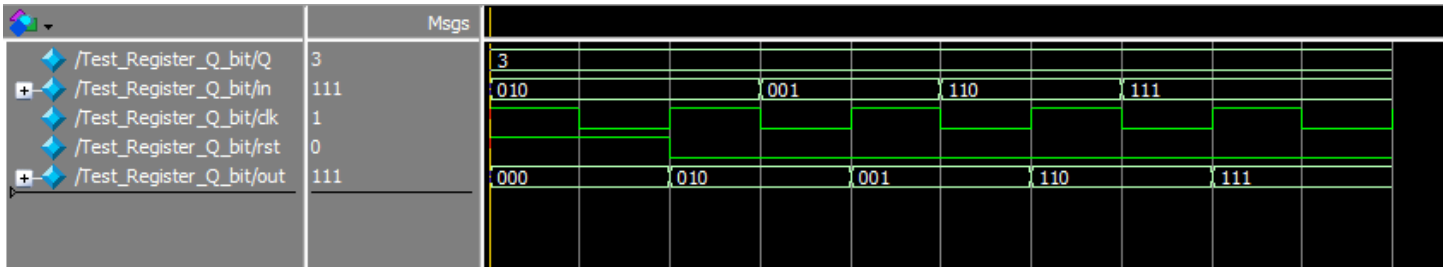
4. Write a behavioral model of a Q-bit Register.




```
1  module Register_Q_bit #(parameter Q = 4)(clk, rst, in, out);
2  input  [Q-1:0]in;
3  input  clk, rst;
4  output reg [Q-1:0]out;
5
6  // behavioral model of a Q bit Register
7  always @(posedge clk, posedge rst)
8  begin
9      if (rst)
10         out <= 0;
11     else
12         out <= in;
13 end
14 endmodule
```

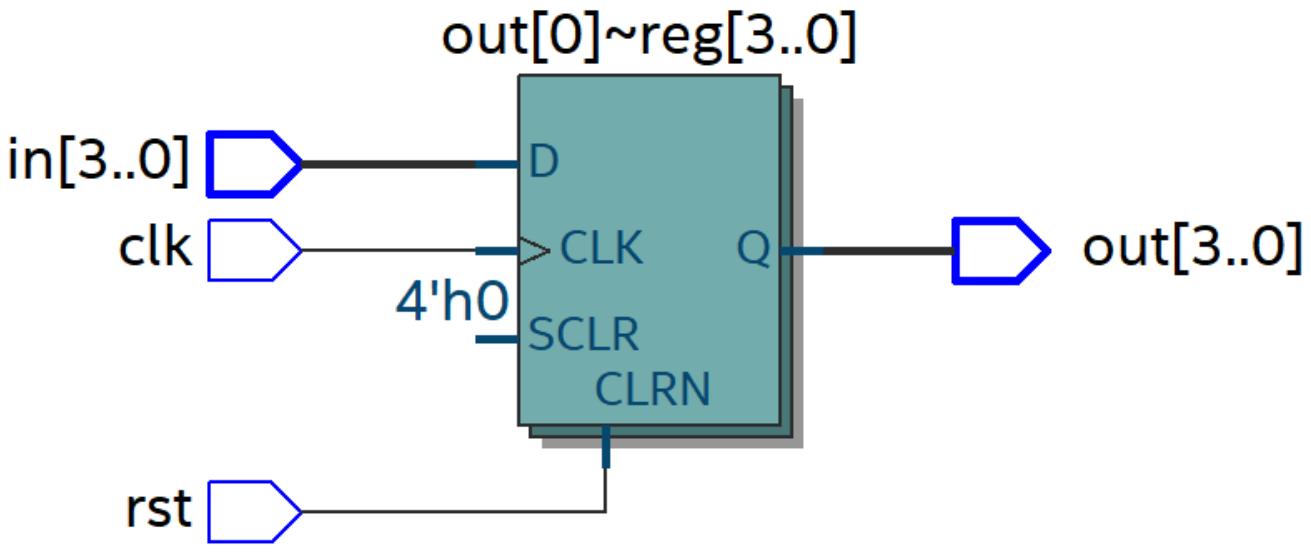
```
1  `timescale 1ps/1ps
2  module Test_Register_Q_bit;
3  parameter Q= 3;
4  reg  [Q-1:0]in;
5  reg  clk, rst;
6  wire [Q-1:0]out;
7
8
9  Register_Q_bit #(Q)RG(clk, rst, in, out);
10
11 initial begin
12     clk =1;
13     forever #50  clk =~clk;
14 end
15
16 initial begin
17     rst =1;
18     #100  rst =0; // rst in one clock cycle
19 end
20
21 initial begin
22 // the change in input will be at the negative edge ,
23 //which can be detected at the next posetve edge of the clock
24     in=2; #150
25     in=1; #100
26     in=6; #100
27     in=7; #100;
28 end
29
30 initial begin
31 $monitor ("time: %3d   in: %b   out: %b ",
32          $time       , in     ,  out       );
33 end
34 endmodule
```

The simulation of the fourth module

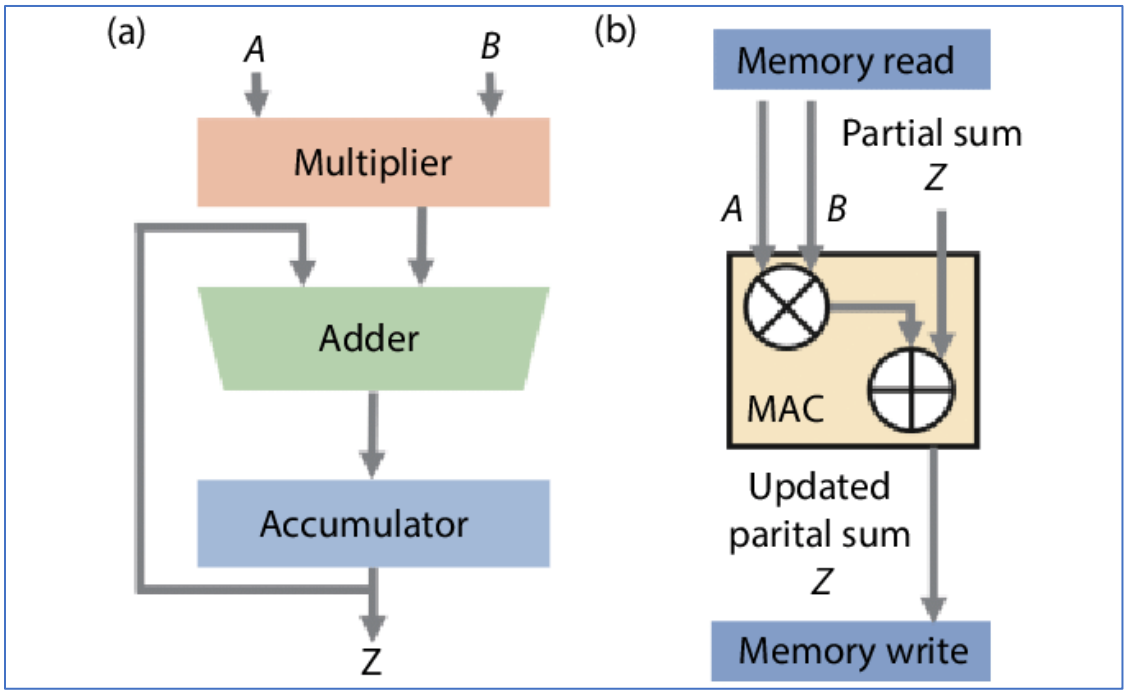



```
# time:   0   in: 010   out: 000
# time: 100   in: 010   out: 010
# time: 150   in: 001   out: 010
# time: 200   in: 001   out: 001
# time: 250   in: 110   out: 001
# time: 300   in: 110   out: 110
# time: 350   in: 111   out: 110
# time: 400   in: 111   out: 111
```

The RTL of the fourth module



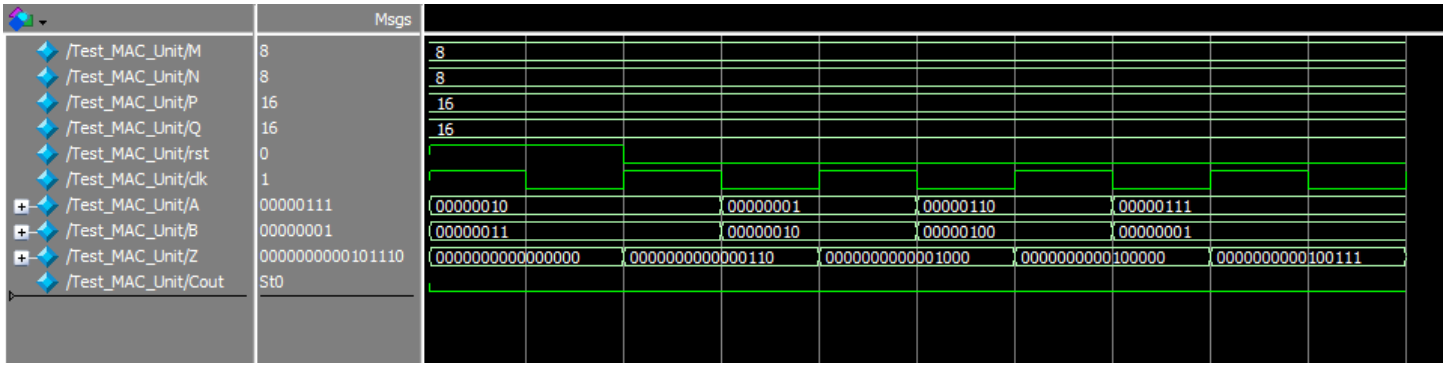
5. Write a structural model of a MAC Unit.




```
1  module MAC_Unit#(parameter M=8, N=8, P=16, Q=16)(clk , rst, A, B, Cout, Z);
2
3  input  rst, clk;
4  input  [M-1:0]A;
5  input  [N-1:0]B;
6  output [M+N-1:0]Z;
7  output Cout;
8  wire   [M+N-1:0]Mul;   // the out of multiplication operation
9  wire   [M+N-1:0]add;   // the out of addition
10
11 // Structural model of a MAC Unit
12 Multiplier_M_N_bits #(M,N) Multp (A, B, Mul);
13 Full_Adder_P_bit    #(P)   Adder (Mul, Z, 1'b0, add, Cout);
14 Register_Q_bit      #(Q)   RegBit(clk, rst, add, Z);
15 endmodule
```

```
1  `timescale 1ps/1ps
2  module Test_MAC_Unit;
3  parameter M = 8, N= 8, P= 16, Q= 16;
4  reg  rst, clk;
5  reg  [M-1:0]A;
6  reg  [N-1:0]B;
7  wire [M+N-1:0]Z;
8  wire Cout;
9
10 MAC_Unit#(M, N, P, Q)MAC(clk , rst, A, B, Cout, Z);
11 initial begin
12     clk =1;
13     forever #50  clk =~clk;
14 end
15 initial begin
16     rst =1;
17     #100  rst =0; // rst in one clock cycle
18 end
19 initial begin
20 // the change in input will be at the negative edge ,
21 //which can be detected at the next posetve edge of the clock
22     A=2; B=3; #150
23     A=1; B=2; #100
24     A=6; B=4; #100
25     A=7; B=1; #100;
26 end
27 initial begin
28 $monitor ("time: %3d   A: %b   B: %b   Z: %b ",
29          $time      , A    ,  B    ,  Z     );
30 end
31 endmodule
```

The simulation of the top module




```
# time:   0   A: 00000010   B: 00000011   Z: 0000000000000000
# time: 100   A: 00000010   B: 00000011   Z: 0000000000000110
# time: 150   A: 00000001   B: 00000010   Z: 0000000000000110
# time: 200   A: 00000001   B: 00000010   Z: 0000000000001000
# time: 250   A: 00000110   B: 00000100   Z: 0000000000001000
# time: 300   A: 00000110   B: 00000100   Z: 0000000000100000
# time: 350   A: 00000111   B: 00000001   Z: 0000000000100000
# time: 400   A: 00000111   B: 00000001   Z: 0000000000100111
```

The RTL of the top module