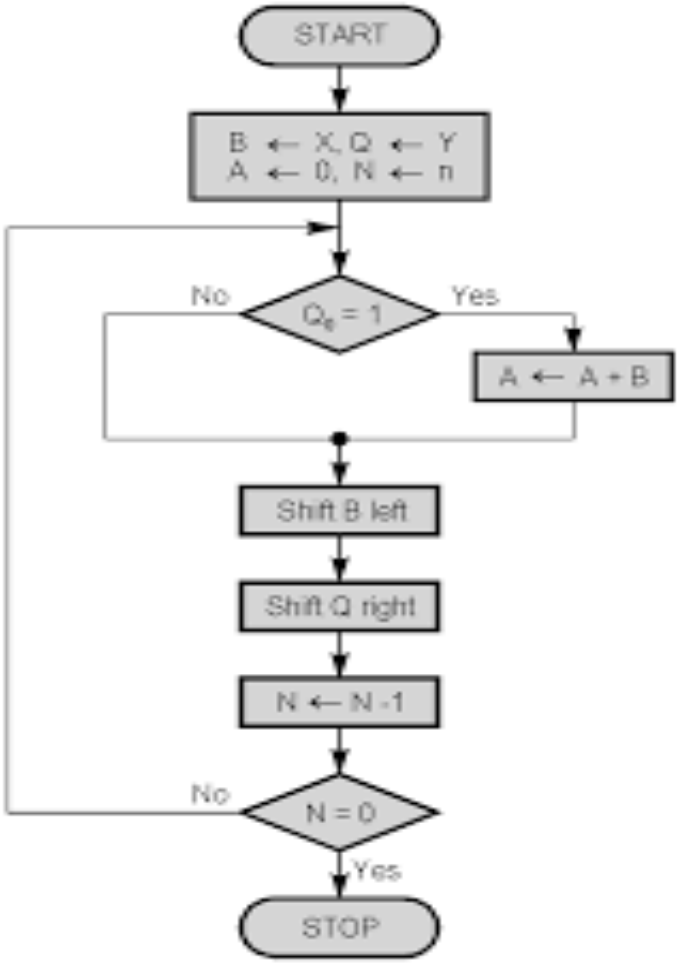
Q.2. Implement a 8-bit multiplier which makes use of a 2-bit full adder (shift-add algorithm).

Algorithm



Main Code

```
module fulladder_2bit (  input [1:0] a,input [1:0] b,input c_in,output reg [1:0] sum,output reg c_out);


    always @ (a,b,c_in) begin
    {c_out, sum} = a + b + c_in;
  end
endmodule

//module to create 8 bit full adder using 2 bit full adder
module fulladder_8bit_1(input [7:0] a,input [7:0] b,input c_in,output reg [7:0] sum,output reg c_out,input clk);
wire c1,c2,c3,cout_temp;
wire [7:0] sum_temp;
fulladder_2bit a1(c1,sum_temp[1:0],a[1:0],b[1:0],c_in);
fulladder_2bit a2(c2,sum_temp[3:2],a[3:2],b[3:2],c1);
fulladder_2bit a3(c3,sum_temp[5:4],a[5:4],b[5:4],c2);
fulladder_2bit a4(c4,sum_temp[7:6],a[7:6],b[7:6],c3);
always@(posedge clk)
begin
    sum<=sum_temp;
    c_out<=cout_temp;
end
endmodule
//module to create 8 bit adder
module fulladder_8bit(  input [7:0] a,input [7:0] b,input c_in,output reg [15:0] sum,output reg c_out);


    always @ (a,b,c_in) begin
    {c_out, sum} = a + b + c_in;
  end
```

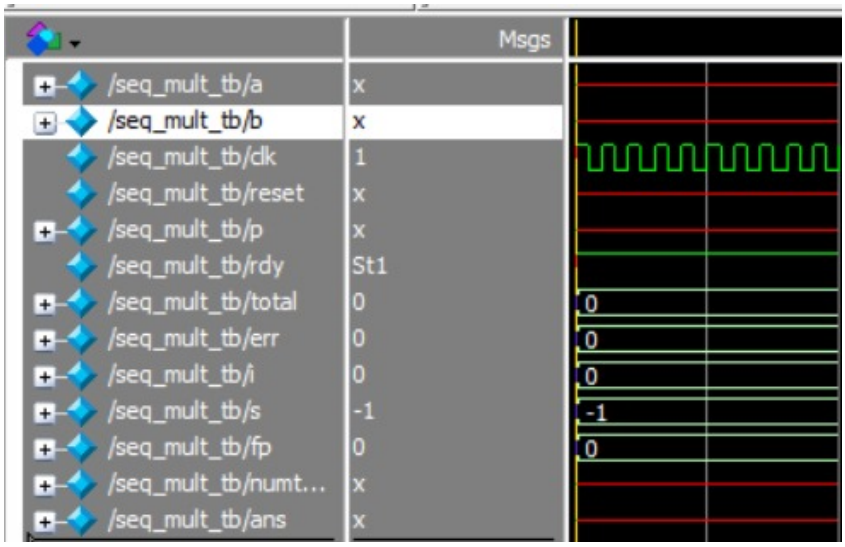
```
endmodule

//implementation of multiplier using shift-add algorithm
module multiplier(input [7:0] a,input [7:0] b, output reg[15:0] product);
wire count = 3'b0;
wire Q = b;
wire A = 8'b0;
wire C = 1'b0;
wire [16:0] partial_result = {C,A,Q};
wire carry = 1'b0;
reg [7:0] temp_sum;
wire temp_cout;


always @(*) begin
    while(count<3'b111)//checking last bit of b input
    //the output is in the form of CAP, where CAP is shifted
    begin
        if(Q[0]==1'b1)
        begin

            fulladder_8bit sum1(A,b,carry,temp_sum,temp_cout);
            A<=temp_sum;
            partial_result = {C,A,Q};
        end
        partial_result = partial_result>>1'b1;
        Q = partial_result[7:0];
        A = partial_result[15:8];
        C = partial_result[16];
```

```
        count = count+1;
    end
end
assign product = partial_result[15:0];
endmodule
```

Test Bench Code

```
`timescale 1ns / 1ps
module multiplier(input [7:0] a,input [7:0] b, output reg[15:0] product);
eg clk;
reg reset;
// Design Inputs and outputs
reg A[8:0];
reg B{8:0];
reg out[15:0];
endmodule
module multiplier_tb ();

  // Clock and reset signals

initial begin
    // Use the monitor task to display the FPGA IO

    $monitor("time=%3d, in_a=%b, in_b=%b, q=%2b \n",$time, in_a, in_b, q);

     // Generate each input with a 20 ns delay between them
    in_a = 1'b0;
    in_b = 1'b0;
    #20
    in_a = 1'b1;
    #20
    in_a = 1'b0;
    in_b = 1'b1;
    #20
    in_a = 1'b1;
 end
endmodule;
```

Output


Msgs
/seq_mult_tb/a
/seq_mult_tb/b
/seq_mult_tb/clk
/seq_mult_tb/reset
/seq_mult_tb/p
/seq_mult_tb/rdy
/seq_mult_tb/total
/seq_mult_tb/err
/seq_mult_tb/i
/seq_mult_tb/s
/seq_mult_tb/fp
/seq_mult_tb/numt...
/seq_mult_tb/ans
x
x
1
x
x
St1
0
0
0
-1
0
x
x
0
0
0
-1
0