Q1)

```
module arithmetic_shift_register (
    input wire clk,              // Clock input
    input wire load,             // Synchronous load enable
    input wire ena,              // Shift enable
    input wire [1:0] amount,     // Shift direction and amount (00: LSL 1, 01: LSL 8, 10: ASR 1, 11: ASR 8)
    input wire [63:0] data,      // 64-bit data input for load
    output reg [63:0] q          // 64-bit shift register output
);

    always @(posedge clk) begin
        if (load) begin
            // Synchronous load: load data into q
            q <= data;
        end else if (ena) begin
            // Shift operation based on amount
            case (amount)
                2'b00: q <= q << 1; //{q[62:0], 1'b0};                  // Logical left shift by 1
                2'b01: q <= q << 8; //{q[55:0], 8'b0};                  // Logical left shift by 8
                2'b10: q <= {q[63], q[63:1]};                  // Arithmetic right shift by 1
                2'b11: q <= {{8{q[63]}}, q[63:8]};             // Arithmetic right shift by 8
                default: q <= q;                               // No shift (redundant, for completeness)
            endcase
        end
        // If neither load nor ena is asserted, q retains its value
    end

endmodule
```

```
module arithmetic_shift_register_tb;

    // Testbench signals
    reg clk;                    // Clock
    reg load;                   // Load enable
    reg ena;                    // Shift enable
    reg [1:0] amount;           // Shift direction and amount
    reg  [63:0] data;            // Input data
    wire [63:0] q;       // Shift register output

    // Instantiate the DUT (Device Under Test)
    arithmetic_shift_register dut (
        .clk(clk),
        .load(load),
        .ena(ena),
        .amount(amount),
        .data(data),
        .q(q)
    );

    // Clock generation: 10ns period (5ns high, 5ns low)
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    // Test stimulus
    initial begin
        // Initialize signals
        load = 0;
        ena = 0;
        amount = 2'b00;
        data = 64'h0;
        @(negedge clk);
```

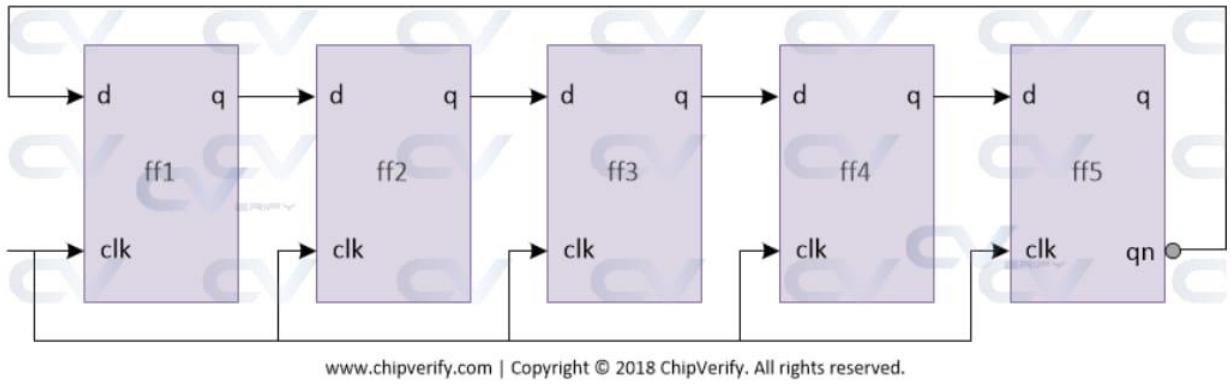
```
        // Test 1: Load a positive value
        load = 1;
        data = 64'h0000_0000_1234_5678;
        @(negedge clk);
        load = 0;
        $display("Time=%0t: After load positive, q=%h", $time, q);

        // Test 2: Left shift by 1
        ena = 1;
        amount = 2'b00; // LSL 1
        @(negedge clk);
        ena = 0;
        $display("Time=%0t: After LSL 1, q=%h", $time, q);

        // Test 3: Left shift by 8
        load = 1;
        data = 64'h0000_0000_1234_5678;
        @(negedge clk);
        load = 0;
        ena = 1;
        amount = 2'b01; // LSL 8
        @(negedge clk);
        ena = 0;
        $display("Time=%0t: After LSL 8, q=%h", $time, q);
```

```
        // Test 4: Load a negative value (sign bit = 1)
        load = 1;
        data = 64'hFFFF_FFFF_8765_4321;
        @(negedge clk);
        load = 0;
        $display("Time=%0t: After load negative, q=%h", $time, q);

        // Test 5: Arithmetic right shift by 1
        ena = 1;
        amount = 2'b10; // ASR 1
        @(negedge clk);         ena = 0;
        $display("Time=%0t: After ASR 1, q=%h", $time, q);

        // Test 6: Arithmetic right shift by 8
        load = 1;
        data = 64'hFFFF_FFFF_8765_4321;
        @(negedge clk);
        load = 0;
        ena = 1;
        amount = 2'b11; // ASR 8
        @(negedge clk);
        ena = 0;
        $display("Time=%0t: After ASR 8, q=%h", $time, q);

        // Finish simulation
        @(negedge clk);       $display("Simulation completed");
        $stop;
    end

endmodule
```

Q2)

```
module johnson_counter_param #(
    parameter WIDTH = 4  // Default width is 4 bits
) (
    input wire clk,         // Clock input
    input wire reset,       // Active-high reset
    output reg [WIDTH-1:0] Q  // Counter output
);

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            // Reset to all zeros
            Q <= {WIDTH{1'b0}};
        end else begin
            Q <= {Q[WIDTH-2:0], ~Q[WIDTH-1]};

            // or you can write it like this
            //Q[0] <= ~Q[WIDTH-1];
            //Q[WIDTH-1:1] <= Q[WIDTH-2:0];

        end
    end

endmodule
```

For better visualization



```
module johnson_counter_param_tb;
    // Testbench signals
    reg clk;                // Clock
    reg reset;              // Reset
    wire [3:0] Q;           // 4-bit counter output

    // Instantiate the DUT with WIDTH=4
    johnson_counter_param #(.WIDTH(4)) dut (.clk(clk),.reset(reset),.Q(Q));

    // Clock generation: 10ns period
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    // Test stimulus
    initial begin
        // Initialize signals
        reset = 1;
        @(negedge clk);
        reset = 0; // Release reset after 10ns

        // Run for 10 clock cycles to observe the sequence and repeat
        repeat(10) begin
            @(negedge clk); // 10 cycles * 10ns = 100ns
        end

        $display("Test completed");
        $stop;
    end

    // Monitor outputs
    initial begin
        $display("Time\tReset\tQ[3:0]");
        $monitor("%0t\t%b\t%b", $time, reset, Q);
    end
```

Q3)

```
1   module Assignment_4_Q3 (A, B, out);
2
3       parameter N = 8;
4       parameter TYPE = 0;
5
6       input [N-1:0] A,B;
7       output [(2*N)-1:0] out;
8
9       generate
10          if (!TYPE) begin
11              Adder dut0 (A, B, out);
12          end
13          else if (TYPE) begin
14              Multiplier dut1 (A, B, out);
15          end
16      endgenerate
17
18  endmodule
```

```
1   module Adder (A, B, out);
2
3       parameter N = 8;
4
5       input [N-1:0] A,B;
6       output [(2*N)-1:0] out;
7
8       assign out=A+B;
9
10  endmodule
```

```
1   module Multiplier (A, B, out);
2
3       parameter N = 8;
4
5       input [N-1:0] A,B;
6       output [(2*N)-1:0] out;
7
8       assign out=A*B;
9
10  endmodule
```

```
module Assignment_4_Q3_tb ();   //this is a normal combinational testbench
// you can do this or take each part alone (TYPE=0, TYPE=1)
    parameter N = 8;
    parameter TYPE = 0; // try "0" first then "1"

    reg [N-1:0] A,B;
    wire [(2*N)-1:0] out;

    Assignment_4_Q3 #(.TYPE(TYPE)) dut (A, B, out);

    initial begin
        if (TYPE==0) begin
            A=5; B=5;
            #10 //(because its combinational not seq.)
            if (out!=10) begin
                $display("error");
                $stop;
            end
            A=5; B=10;
            #10
            if (out!=15) begin
                $display("error");
                $stop;
            end
        end
        else if (TYPE==1) begin
            A=5; B=5;
            #10
            if (out!=25) begin
                $display("error");
                $stop;
            end
            A=5; B=10;
            #10
            if (out!=50) begin
                $display("error");
                $stop;
            end
        end
        $stop;
    end

endmodule : Assignment_4_Q3_tb
```