1. Implement the 4-bit Adder-Subtractor Circuit using Verilog

Module:

```
// 4-bit Adder-Subtractor Module
module AdderSubtractor4Bit(
    input [3:0] A,    // First 4-bit operand
    input [3:0] B,    // Second 4-bit operand
    input mode,       // Mode: 0 for addition, 1 for subtraction
    output [3:0] Sum, // 4-bit Result
    output CarryOut   // Carry or Borrow Out
);
    wire [3:0] B_xor;  // XOR of B with mode
    wire CarryIn;      // Carry-In for LSB
    wire [3:0] Carry;  // Internal carry wires

    // XOR the B input with the mode signal
    assign B_xor = B ^ {4{mode}};

    // Perform bitwise addition with carry propagation
    assign {CarryOut, Sum} = A + B_xor + mode;
endmodule
```

Test Bench

```
`timescale 1ns / 1ps

module AdderSubtractor4Bit_tb;


    reg [3:0] A;          // First operand
    reg [3:0] B;          // Second operand
    reg mode;             // Mode input: 0 for addition, 1 for subtraction
    wire [3:0] Sum;       // Output result
    wire CarryOut;        // Carry/Borrow out

    // Instantiate the design under test (DUT)
    AdderSubtractor4Bit uut (
        .A(A),
        .B(B),
        .mode(mode),
        .Sum(Sum),
        .CarryOut(CarryOut)
    );
```

```
    // Test sequence
    initial begin
        // Display the header
        $display("Time | A    | B    | Mode | Sum  | CarryOut");
        $display("-----------------------------------------------");

        // Test cases
        A = 4'b0001; B = 4'b0010; mode = 0; #10; // Addition: 1 + 2
        $display("%4t | %b | %b | %b    | %b | %b", $time, A, B, mode, Sum, CarryOut);

        A = 4'b1010; B = 4'b0101; mode = 0; #10; // Addition: 10 + 5
        $display("%4t | %b | %b | %b    | %b | %b", $time, A, B, mode, Sum, CarryOut);

        A = 4'b1111; B = 4'b0001; mode = 0; #10; // Addition with CarryOut: 15 + 1
        $display("%4t | %b | %b | %b    | %b | %b", $time, A, B, mode, Sum, CarryOut);

        A = 4'b1001; B = 4'b0011; mode = 1; #10; // Subtraction: 9 - 3
        $display("%4t | %b | %b | %b    | %b | %b", $time, A, B, mode, Sum, CarryOut);

        A = 4'b0100; B = 4'b1000; mode = 1; #10; // Subtraction with BorrowOut: 4 - 8
        $display("%4t | %b | %b | %b    | %b | %b", $time, A, B, mode, Sum, CarryOut);

        A = 4'b0000; B = 4'b0000; mode = 1; #10; // Subtraction: 0 - 0
        $display("%4t | %b | %b | %b    | %b | %b", $time, A, B, mode, Sum, CarryOut);

        $stop; // End simulation
    end
endmodule
```

2. Implement the BCD to Excess-3 code converter Circuit using verilog with test-bench

Module

```
// BCD to Excess-3 Code Converter
module BCDtoExcess3(
    input [3:0] BCD,       // 4-bit BCD input
    output [3:0] Excess3   // 4-bit Excess-3 output
);
    // Add 3 to the BCD input to convert to Excess-3
    assign Excess3 = BCD + 4'b0011;
endmodule
```

Test_bench:

```
`timescale 1ns / 1ps

module BCDtoExcess3_tb;

    reg [3:0] BCD;          // 4-bit BCD input
    wire [3:0] Excess3;     // 4-bit Excess-3 output

    // Instantiate the design under test (DUT)
    BCDtoExcess3 uut (
        .BCD(BCD),
        .Excess3(Excess3)
    );

    // Test sequence
    initial begin
        // Display header
        $display("Time | BCD  | Excess-3");
        $display("----------------------");

        // Test cases
        BCD = 4'b0000; #10; // 0 -> 3
        $display("%4t | %b | %b", $time, BCD, Excess3);

        BCD = 4'b0001; #10; // 1 -> 4
        $display("%4t | %b | %b", $time, BCD, Excess3);

        BCD = 4'b0010; #10; // 2 -> 5
        $display("%4t | %b | %b", $time, BCD, Excess3);

        BCD = 4'b0011; #10; // 3 -> 6
        $display("%4t | %b | %b", $time, BCD, Excess3);

        BCD = 4'b0100; #10; // 4 -> 7
        $display("%4t | %b | %b", $time, BCD, Excess3);

        BCD = 4'b0101; #10; // 5 -> 8
        $display("%4t | %b | %b", $time, BCD, Excess3);

        BCD = 4'b0110; #10; // 6 -> 9
        $display("%4t | %b | %b", $time, BCD, Excess3);
```

```
        BCD = 4'b0111; #10; // 7 -> 10
        $display("%4t | %b | %b", $time, BCD, Excess3);

        BCD = 4'b1000; #10; // 8 -> 11
        $display("%4t | %b | %b", $time, BCD, Excess3);

        BCD = 4'b1001; #10; // 9 -> 12
        $display("%4t | %b | %b", $time, BCD, Excess3);

        // Invalid BCD values (optional testing)
        BCD = 4'b1010; #10; // Invalid -> 13
        $display("%4t | %b | %b", $time, BCD, Excess3);

        $stop; // End simulation
    End
endmodule
```

3. Implement the Excess-3 code to BCD code converter Circuit using Verilog and test-bench

Module

```
// Excess-3 Code to BCD Code Converter
module Excess3toBCD(
    input [3:0] Excess3,  // 4-bit Excess-3 input
    output [3:0] BCD      // 4-bit BCD output
);
    // Subtract 3 from the Excess-3 input to convert to BCD
    assign BCD = Excess3 - 4'b0011;
endmodule
```

Test_bench

```
`timescale 1ns / 1ps
module Excess3toBCD_tb;

    reg [3:0] Excess3;      // 4-bit Excess-3 input
    wire [3:0] BCD;         // 4-bit BCD output

    // Instantiate the design under test (DUT)
    Excess3toBCD uut (
        .Excess3(Excess3),
        .BCD(BCD)
    );
```

```
// Test sequence
initial begin
    // Display header
    $display("Time | Excess-3 | BCD");
    $display("--------------------");

    // Test cases
    Excess3 = 4'b0011; #10; // Excess-3: 3 -> BCD: 0
    $display("%4t | %b     | %b", $time, Excess3, BCD);

    Excess3 = 4'b0100; #10; // Excess-3: 4 -> BCD: 1
    $display("%4t | %b     | %b", $time, Excess3, BCD);

    Excess3 = 4'b0101; #10; // Excess-3: 5 -> BCD: 2
    $display("%4t | %b     | %b", $time, Excess3, BCD);

    Excess3 = 4'b0110; #10; // Excess-3: 6 -> BCD: 3
    $display("%4t | %b     | %b", $time, Excess3, BCD);

    Excess3 = 4'b0111; #10; // Excess-3: 7 -> BCD: 4
    $display("%4t | %b     | %b", $time, Excess3, BCD);

    Excess3 = 4'b1000; #10; // Excess-3: 8 -> BCD: 5
    $display("%4t | %b     | %b", $time, Excess3, BCD);

    Excess3 = 4'b1001; #10; // Excess-3: 9 -> BCD: 6
    $display("%4t | %b     | %b", $time, Excess3, BCD);

    Excess3 = 4'b1010; #10; // Excess-3: 10 -> BCD: 7
    $display("%4t | %b     | %b", $time, Excess3, BCD);

    Excess3 = 4'b1011; #10; // Excess-3: 11 -> BCD: 8
    $display("%4t | %b     | %b", $time, Excess3, BCD);

    Excess3 = 4'b1100; #10; // Excess-3: 12 -> BCD: 9
    $display("%4t | %b     | %b", $time, Excess3, BCD);

    // Invalid Excess-3 values (optional testing)
    Excess3 = 4'b1101; #10; // Invalid Excess-3
    $display("%4t | %b     | %b", $time, Excess3, BCD);

    $stop; // End simulation
```

```
    End
endmodule
```

4. Implement the 4-bit Asynchronous Mod-13 counter using verilog and Test-bench

Module

```
// 4-bit Asynchronous Mod-13 Counter
module Mod13Counter(
    input clk,           // Clock input
    input reset,         // Asynchronous reset
    output reg [3:0] Q    // 4-bit counter output
);

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            Q <= 4'b0000; // Reset the counter to 0
        end else begin
            if (Q == 4'b1100) // If the counter reaches 13 (1100 in binary)
                Q <= 4'b0000; // Reset to 0
            else
                Q <= Q + 1;   // Increment the counter
        end
    end

endmodule
```

Test_bench

```
`timescale 1ns / 1ps

module Mod13Counter_tb;

    reg clk;             // Clock input
    reg reset;           // Asynchronous reset
    wire [3:0] Q;        // 4-bit counter output

    // Instantiate the design under test (DUT)
    Mod13Counter uut (
        .clk(clk),
        .reset(reset),
```

```
    .Q(Q)
);

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

// Test sequence
initial begin
    // Display header
    $display("Time | Reset | Q");
    $display("----------------");

    // Apply reset
    reset = 1; #10;
    reset = 0; #10;

    // Let the counter run
    #130; // Run for 130 ns (to see multiple cycles)

    // Apply reset during counting
    reset = 1; #10;
    reset = 0; #10;

    // Let the counter run again
    #100;

    $stop; // End simulation
end

// Monitor output
initial begin
    $monitor("%4t | %b     | %b", $time, reset, Q);
end

endmodule
```

5. Implement the 3-bit up/down counter using verilog with test-bench

Module

```
// 3-bit Up/Down Counter
module UpDownCounter(
    input clk,            // Clock input
    input reset,          // Asynchronous reset
    input up_down,        // Direction control: 1 for up, 0 for down
    output reg [2:0] Q    // 3-bit counter output
);

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            Q <= 3'b000; // Reset counter to 0
        end else begin
            if (up_down)
                Q <= Q + 1; // Count up
            else
                Q <= Q - 1; // Count down
        end
    end
endmodule
```

Test_bench

```
`timescale 1ns / 1ps
module UpDownCounter_tb;

    reg clk;            // Clock input
    reg reset;          // Asynchronous reset
    reg up_down;        // Direction control
    wire [2:0] Q;       // 3-bit counter output
```

```
// Instantiate the design under test (DUT)
UpDownCounter uut (
    .clk(clk),
    .reset(reset),
    .up_down(up_down),
    .Q(Q)
);

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

// Test sequence
initial begin
    // Display header
    $display("Time | Reset | Up/Down | Q");
    $display("------------------------");

    // Apply reset
    reset = 1; up_down = 1; #10; // Reset the counter
    reset = 0; #10;

    // Count up
    up_down = 1; #50; // Let it count up for 50 ns

    // Count down
```

```
        up_down = 0; #50; // Let it count down for 50 ns

        // Apply reset during operation
        reset = 1; #10;
        reset = 0; #10;

        // Count up again
        up_down = 1; #30;

        $stop; // End simulation
    end

    // Monitor output
    initial begin
        $monitor("%4t | %b     | %b        | %b", $time, reset, up_down, Q);
    end
endmodule
```