8-bit ripple carry adder using 1-bit full adder

VERILOG-

```
module full_adder (
    input  wire a,
    input  wire b,
    input  wire cin,
    output wire sum,
    output wire cout
);
    assign sum  = a ^ b ^ cin;
    assign cout = (a & b) | (b & cin) | (a & cin);
endmodule

module ripple_carry_adder_8bit (
    input  wire [7:0] a,
    input  wire [7:0] b,
    input  wire cin,
    output wire [7:0] sum,
    output wire cout
);
    wire [6:0] carry;

    full_adder fa0 (a[0], b[0], cin,       sum[0], carry[0]);
    full_adder fa1 (a[1], b[1], carry[0],   sum[1], carry[1]);
    full_adder fa2 (a[2], b[2], carry[1],   sum[2], carry[2]);
    full_adder fa3 (a[3], b[3], carry[2],   sum[3], carry[3]);
    full_adder fa4 (a[4], b[4], carry[3],   sum[4], carry[4]);
    full_adder fa5 (a[5], b[5], carry[4],   sum[5], carry[5]);
    full_adder fa6 (a[6], b[6], carry[5],   sum[6], carry[6]);
    full_adder fa7 (a[7], b[7], carry[6],   sum[7], cout);
endmodule
```

54

TESTBENCH-

```
`timescale 1ns / 1ps

module tb_ripple_carry_adder_8bit;
    reg  [7:0] a, b;
    reg        cin;
    wire [7:0] sum;
    wire       cout;

    ripple_carry_adder_8bit uut (
        .a(a), .b(b), .cin(cin),
        .sum(sum), .cout(cout)
    );

    initial begin
        $monitor("Time=%0t a=%b b=%b cin=%b => sum=%b cout=%b", $time, a, b, cin, sum,
cout);

        a = 8'b00000000; b = 8'b00000000; cin = 0; #10;
        a = 8'b00001111; b = 8'b00000001; cin = 0; #10;
        a = 8'b11111111; b = 8'b00000001; cin = 0; #10;
        a = 8'b10101010; b = 8'b01010101; cin = 0; #10;
        a = 8'b11110000; b = 8'b00001111; cin = 1; #10;
        a = 8'b11111111; b = 8'b11111111; cin = 1; #10;


        $finish;
    end
endmodule
```

----

8 bit bidirectional shifter

verilog -

```
module shifter_8bit (
    input  wire [7:0] data_in,
    input  wire [2:0] shift_amt,  // up to 7-bit shift
    input  wire       dir,        // 0 = left shift, 1 = right shift
    output wire [7:0] data_out
);
    assign data_out = (dir == 1'b0) ? (data_in << shift_amt) : (data_in >> shift_amt);
endmodule
```

testbench-

```
`timescale 1ns / 1ps


module tb_shifter_8bit;
    reg  [7:0] data_in;
    reg  [2:0] shift_amt;
    reg        dir;
    wire [7:0] data_out;
    shifter_8bit uut (
        .data_in(data_in),
        .shift_amt(shift_amt),
        .dir(dir),
        .data_out(data_out)
    );

    initial begin
        $monitor("Time=%0t data_in=%b shift_amt=%d dir=%b => data_out=%b", $time,
data_in, shift_amt, dir, data_out);

        // Test left shifts
        data_in = 8'b00001111; shift_amt = 3'd1; dir = 0; #10;
        data_in = 8'b00001111; shift_amt = 3'd2; dir = 0; #10;
        data_in = 8'b10000001; shift_amt = 3'd3; dir = 0; #10;

        // Test right shifts
        data_in = 8'b11110000; shift_amt = 3'd1; dir = 1; #10;
        data_in = 8'b11110000; shift_amt = 3'd2; dir = 1; #10;
        data_in = 8'b10000001; shift_amt = 3'd3; dir = 1; #10;

        $finish;
    end
endmodule
```

----

8 bit magnitude comparator

verilog-

```
module magnitude_comparator_8bit (
    input  wire [7:0] a,
    input  wire [7:0] b,
    output wire       a_gt_b,
    output wire       a_lt_b,
    output wire       a_eq_b
);

assign a_gt_b = (a > b);
assign a_lt_b = (a < b);
assign a_eq_b = (a == b);

endmodule
```

testbench-

```
`timescale 1ns / 1ps

module tb_magnitude_comparator_8bit;
    reg  [7:0] a, b;
    wire       a_gt_b, a_lt_b, a_eq_b;

    magnitude_comparator_8bit uut (
        .a(a),
        .b(b),
        .a_gt_b(a_gt_b),
        .a_lt_b(a_lt_b),
        .a_eq_b(a_eq_b)
    );

    initial begin
        $monitor("Time=%0t A=%b B=%b | A>B=%b A<B=%b A==B=%b", $time, a, b, a_gt_b,
a_lt_b, a_eq_b);

        a = 8'd100; b = 8'd50;  #10;
        a = 8'd25;  b = 8'd100; #10;
        a = 8'd77;  b = 8'd77;  #10;
        a = 8'd0;   b = 8'd255; #10;
        a = 8'd255; b = 8'd0;   #10;

        $finish;
    end
endmodule
```

3:8 decoder

```
verilog-
module decoder_3to8 (
    input  wire [2:0] in,   // 3-bit input
    input  wire       en,   // Enable signal
    output wire [7:0] out   // 8 output lines
);

assign out = en ? (8'b00000001 << in) : 8'b00000000;
```

```
endmodule
```

testbench-

```
`timescale 1ns / 1ps

module tb_decoder_3to8;
   reg  [2:0] in;
   reg       en;
   wire [7:0] out;

   decoder_3to8 uut (
      .in(in),
      .en(en),
      .out(out)
   );

   initial begin
      $monitor("Time=%0t Enable=%b In=%b => Out=%b", $time, en, in, out);

      en = 1;
      in = 3'd0; #10;
      in = 3'd1; #10;
      in = 3'd2; #10;
      in = 3'd3; #10;
      in = 3'd4; #10;
      in = 3'd5; #10;
      in = 3'd6; #10;
```

```
        in = 3'd7; #10;

        // Disable the decoder
        en = 0; in = 3'd3; #10;

        $finish;
    end
endmodule
```


8:1 mux

verilog-

```
module mux8x1 (
    input  wire [7:0] d0,  // 8-bit input 0
    input  wire [7:0] d1,  // 8-bit input 1
    input  wire [7:0] d2,  // 8-bit input 2
    input  wire [7:0] d3,  // 8-bit input 3
    input  wire [7:0] d4,  // 8-bit input 4
    input  wire [7:0] d5,  // 8-bit input 5
    input  wire [7:0] d6,  // 8-bit input 6
    input  wire [7:0] d7,  // 8-bit input 7
    input  wire [2:0] sel, // 3-bit selector (selects which input)
    output reg  [7:0] y    // 8-bit output
);

    always @(*) begin
```

```
        case (sel)
            3'b000: y = d0;
            3'b001: y = d1;
            3'b010: y = d2;
            3'b011: y = d3;
            3'b100: y = d4;
            3'b101: y = d5;
            3'b110: y = d6;
            3'b111: y = d7;
            default: y = 8'b00000000; // Default case (if needed)
        endcase
    end

endmodule
```

testbench-

```
`timescale 1ns / 1ps

module tb_mux8x1;
    reg [7:0] d0, d1, d2, d3, d4, d5, d6, d7;
    reg [2:0] sel;
    wire [7:0] y;

    mux8x1 uut (
        .d0(d0),
        .d1(d1),
        .d2(d2),
        .d3(d3),
        .d4(d4),
        .d5(d5),
        .d6(d6),
        .d7(d7),
        .sel(sel),
        .y(y)
    );

    initial begin
        // Initialize inputs
        d0 = 8'd10; d1 = 8'd20; d2 = 8'd30; d3 = 8'd40;
        d4 = 8'd50; d5 = 8'd60; d6 = 8'd70; d7 = 8'd80;

        // Test all selectors
        $monitor("Time=%0t sel=%b y=%d", $time, sel, y);

        sel = 3'b000; #10;
        sel = 3'b001; #10;
        sel = 3'b010; #10;
        sel = 3'b011; #10;
        sel = 3'b100; #10;
        sel = 3'b101; #10;
        sel = 3'b110; #10;
        sel = 3'b111; #10;

        $finish;
    end
endmodule
```

1:8 dmux

verilog-

```
module demux1to8 (
   input  wire d,      // 1-bit input
   input  wire [2:0] sel, // 3-bit selector
   output reg [7:0] y   // 8-bit output
);

   always @(*) begin
      // Default case: All outputs low
      y = 8'b00000000;

      // Select the output based on 'sel'
      case (sel)
         3'b000: y[0] = d;
         3'b001: y[1] = d;
         3'b010: y[2] = d;
         3'b011: y[3] = d;
         3'b100: y[4] = d;
         3'b101: y[5] = d;
         3'b110: y[6] = d;
         3'b111: y[7] = d;
         default: y = 8'b00000000; // Safety default
      endcase
   end
```

```
endmodule


testbench-
`timescale 1ns / 1ps


module tb_demux1to8;
    reg d;          // 1-bit input
    reg [2:0] sel; // 3-bit selector
    wire [7:0] y;  // 8-bit output


    demux1to8 uut (
        .d(d),
        .sel(sel),
        .y(y)
    );


    initial begin
        // Monitor the values
        $monitor("Time=%0t sel=%b d=%b y=%b", $time, sel, d, y);


        // Test case where the input is 1
        d = 1;


        // Test each selector value
        sel = 3'b000; #10;
        sel = 3'b001; #10;
        sel = 3'b010; #10;
        sel = 3'b011; #10;

        sel = 3'b100; #10;

        sel = 3'b101; #10;

        sel = 3'b110; #10;

        sel = 3'b111; #10;


        // Test case where the input is 0

        d = 0;

        sel = 3'b000; #10;

        sel = 3'b001; #10;

        sel = 3'b010; #10;

        sel = 3'b011; #10;

        sel = 3'b100; #10;

        sel = 3'b101; #10;

        sel = 3'b110; #10;

        sel = 3'b111; #10;


        $finish;

    end

endmodule
```

----

4 bit up/down counter

verilog-

```
module up_down_counter (

    input  wire clk,        // Clock input

    input  wire rst,        // Reset input (asynchronous)
    input  wire up_down,    // Control: 1 for Up, 0 for Down
    output reg [3:0] count  // 4-bit counter output
);

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            count <= 4'b0000;  // Reset the counter to 0
        end else begin
            if (up_down) begin
                count <= count + 1;  // Count up
            end else begin
                count <= count - 1;  // Count down
            end
        end
    end

endmodule
```

testbench-

```
`timescale 1ns / 1ps

module tb_up_down_counter;
    reg clk;
    reg rst;
    reg up_down;  // Control: 1 for Up, 0 for Down
    wire [3:0] count;

    up_down_counter uut (
        .clk(clk),
        .rst(rst),
        .up_down(up_down),
        .count(count)
    );

    // Clock generation
    initial clk = 0;
    always #5 clk = ~clk;  // Clock period: 10ns

    // Test sequence
    initial begin
        // Initialize signals
        rst = 1; up_down = 1;  // Start with reset and counting up
        #10;
        rst = 0;  // Deassert reset

        // Count up for a few cycles
        up_down = 1; #50;  // Count up (5 clock cycles)

        // Count down for a few cycles
        up_down = 0; #50;  // Count down (5 clock cycles)

        // Reset the counter
        rst = 1; #10;
        rst = 0; #10;

        // Count up again
```

```
        up_down = 1; #50;


        $finish;
    end
endmodule
```

4-bit Adder-cum-subtractor

```
module addsub(
    input [7:0] a,      // 8-bit input a
    input [7:0] b,      // 8-bit input b
    input mode,         // Mode control: 1 for addition, 0 for subtraction
    output [7:0] result, // 8-bit result
    output carrybor     // Carry bit
);

    wire [8:0] temp;   // 9-bit wire to hold the result (including carry-out)

    // Perform addition or subtraction based on the mode
    assign temp = (mode) ? a + b : a - b;

    // Assign the 8-bit result (lower 8 bits of the temp wire)
    assign result = temp[7:0];

    // Carry bit is the 9th bit (carry-out)
    assign carrybor = temp[8];
```

```
endmodule


testbench-


`timescale 1ns / 1ps


module tb_addsub;
   reg [7:0] a, b;    // 8-bit input operands
   reg mode;          // 1 for addition, 0 for subtraction
   wire [7:0] result; // 8-bit result
   wire carrybor;     // Carry-out or borrow bit

   // Instantiate the addsub module
   addsub uut (
      .a(a),
      .b(b),
      .mode(mode),
      .result(result),
      .carrybor(carrybor)
   );

   initial begin
      // Monitor the values for debugging
      $monitor("Time=%0t | a=%d b=%d mode=%b | result=%d carrybor=%b",
               $time, a, b, mode, result, carrybor);

      // Test 1: Addition (15 + 10)
      a = 8'd15; b = 8'd10; mode = 1;  // Add 15 and 10
```

```
    #10;  // Wait for 10 time units

    // Test 2: Subtraction (20 - 10)
    a = 8'd20; b = 8'd10; mode = 0;  // Subtract 10 from 20
    #10;  // Wait for 10 time units

    // Test 3: Subtraction with borrow (30 - 40)
    a = 8'd30; b = 8'd40; mode = 0;  // 30 - 40 (borrow)
    #10;  // Wait for 10 time units

    // Test 4: Addition with overflow (255 + 1)
    a = 8'd255; b = 8'd1; mode = 1;  // 255 + 1 (overflow)
    #10;  // Wait for 10 time units

    // Test 5: Subtraction resulting in negative (0 - 1)
    a = 8'd0; b = 8'd1; mode = 0;  // 0 - 1 (borrow)
    #10;  // Wait for 10 time units

    // Test 6: Addition without carry (10 + 20)
    a = 8'd10; b = 8'd20; mode = 1;  // 10 + 20
    #10;  // Wait for 10 time units

    // Test 7: Subtraction without borrow (50 - 30)
    a = 8'd50; b = 8'd30; mode = 0;  // 50 - 30
    #10;  // Wait for 10 time units

    $finish;  // End simulation
end
```

endmodule

----

8 bit subtractor