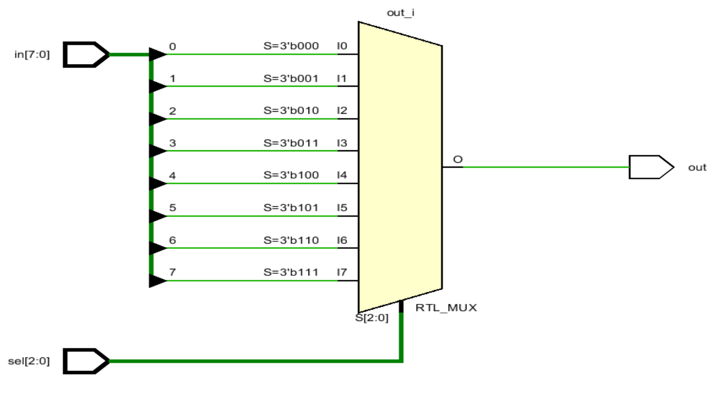
Design a 8x1 Mux using Verilog.

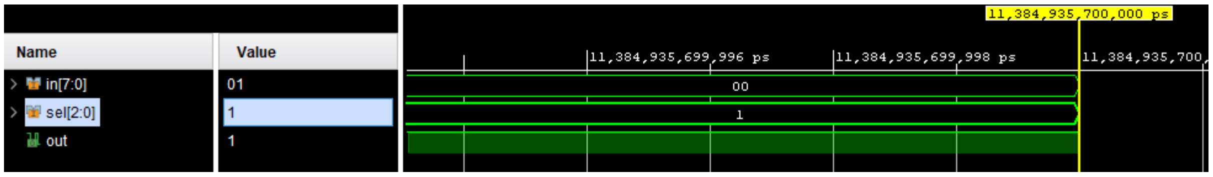
Design code:

```
module mux8x1(
   input [7:0] in,
   input [2:0] sel,
   output reg out
);

always @(*) begin
   case(sel)
      3'b000: out = in[0];
      3'b001: out = in[1];
      3'b010: out = in[2];
      3'b011: out = in[3];
      3'b100: out = in[4];
      3'b101: out = in[5];
      3'b110: out = in[6];
      3'b111: out = in[7];
      default: out = 1'b0;
   endcase
end


endmodule
```



Test Bench:

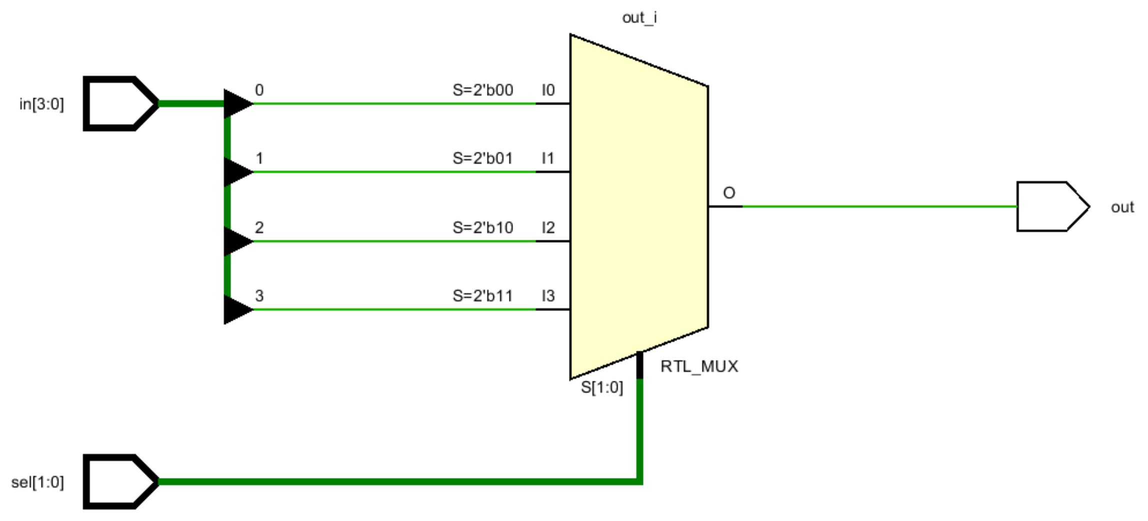
```
`module tb_mux8x1;
   reg [7:0] in;
   reg [2:0] sel;
    wire out;



   mux8x1 uut (
       .in(in),
       .sel(sel),
       .out(out)
   );

   initial begin

       in = 8'b00000000; sel = 3'b000;
       #10 in = 8'b10101010; sel = 3'b000;
       #10 in = 8'b10101010; sel = 3'b001;
       #10 in = 8'b10101010; sel = 3'b010;
       #10 in = 8'b10101010; sel = 3'b011;
       #10 in = 8'b10101010; sel = 3'b100;
       #10 in = 8'b10101010; sel = 3'b101;
       #10 in = 8'b10101010; sel = 3'b110;
       #10 in = 8'b10101010; sel = 3'b111;

       After change:
       #10 in = 8'b11110000; sel = 3'b000;
       #10 in = 8'b11110000; sel = 3'b001;
       #10 in = 8'b11110000; sel = 3'b010;
       #10 in = 8'b11110000; sel = 3'b011;
       #10 in = 8'b11110000; sel = 3'b100
    #10 in = 8'b11110000; sel = 3'b101;

    #10 in = 8'b11110000; sel = 3'b110;

    #10 in = 8'b11110000; sel = 3'b111;



    #10 $finish;

  end



  initial begin

    $monitor("Time=%0d, in=%b, sel=%b, out=%b", $time, in, sel, out);

  end

endmodule
```



Design a 4x1 mux using Verilog hdl

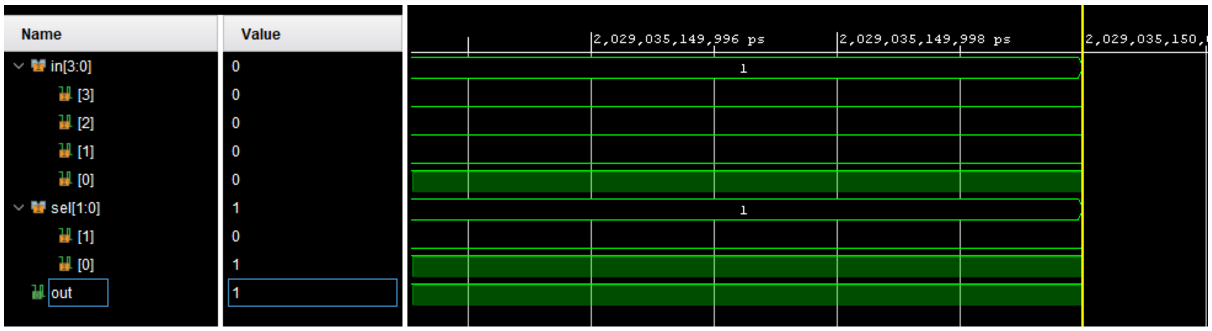
Design code:

```
module mux4x1(
    input [3:0] in,
    input [1:0] sel,
    output reg out
);

always @(*) begin
    case (sel)
        2'b00: out = in[0];
        2'b01: out = in[1];
        2'b10: out = in[2];
        2'b11: out = in[3];
        default: out = 1'b0;
    endcase
end

endmodule
```



Test Bench:

```
module tb_mux4x1;

  reg [3:0] in;

  reg [1:0] sel;

  wire out;

  mux4x1 uut (

    .in(in),

    .sel(sel),

    .out(out)

  );


  initial begin

    in = 4'b0000; sel = 2'b00;


    #10 in = 4'b1010; sel = 2'b00;

    #10 in = 4'b1010; sel = 2'b01;

    #10 in = 4'b1010; sel = 2'b10;

    #10 in = 4'b1010; sel = 2'b11;


    After change:

    #10 in = 4'b1100; sel = 2'b00;

    #10 in = 4'b1100; sel = 2'b01;

    #10 in = 4'b1100; sel = 2'b10;

    #10 in = 4'b1100; sel = 2'b11;


    #10 $finish;

  end

  initial begin

    $monitor("Time=%0d, in=%b, sel=%b, out=%b", $time, in, sel, out);

  end

    endmodule
```



Name	Value
in[3:0]	0
[3]	0
[2]	0
[1]	0
[0]	0
sel[1:0]	1
[1]	0
[0]	1
out	1

Design SISO reg using Verilog hdl

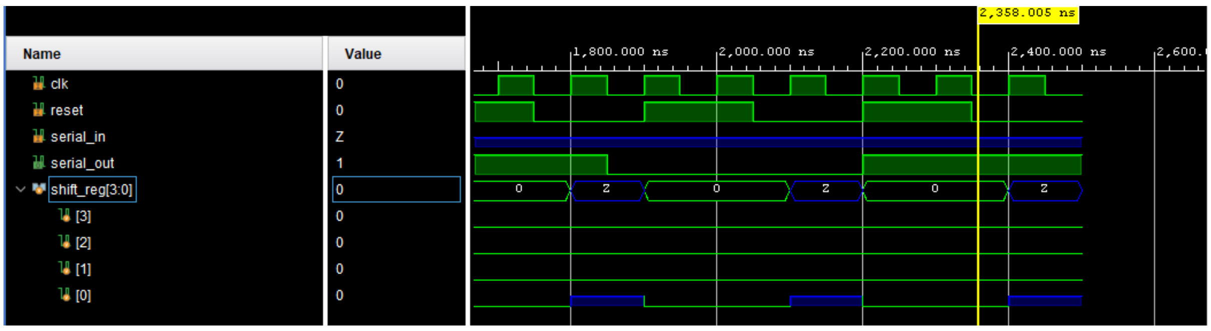
Design code:

```
module siso(
    input wire clk,
    input wire reset,
    input wire serial_in,
    output wire serial_out
);

    reg [3:0] shift_reg;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            shift_reg <= 4'b0000;
        end else begin
            shift_reg <= {shift_reg[2:0], serial_in};
        end
    end

    assign serial_out = shift_reg[3];


endmodule
```

Test Bench:

```
module tb_siso_shift_register;


// Testbench signals

reg clk;                // Clock signal

reg rst;                // Reset signal

reg serial_in;          // Serial input

wire serial_out;        // Serial output


// Instantiate the SISO Shift Register

siso_shift_register uut (

    .clk(clk),

    .rst(rst),

    .serial_in(serial_in),

    .serial_out(serial_out)

);



// Clock generation

initial begin

    clk = 0;

    forever #5 clk = ~clk; // 10 time units clock period
```

```
end

// Test
sequence
initial
begin
    // Initialize signals
    rst = 1;            //
    Activate reset serial_in
    = 0;
    #10;                // Wait for some time

    rst = 0;            // Deactivate reset
    // Apply serial input bits
    serial_in = 1; #10;  //
    Shift in 1
    serial_in = 0; #10;  // Shift in 0
    serial_in = 1; #10;  // Shift in 1
    serial_in = 0; #10;  // Shift in 0
    serial_in = 1; #10;  // Shift in 1
    serial_in = 1; #10;  // Shift in 1
    serial_in = 0; #10;  // Shift in 0
    serial_in = 1; #10;  // Shift in 1


    #10;
    $stop;

    end

endmodule
```

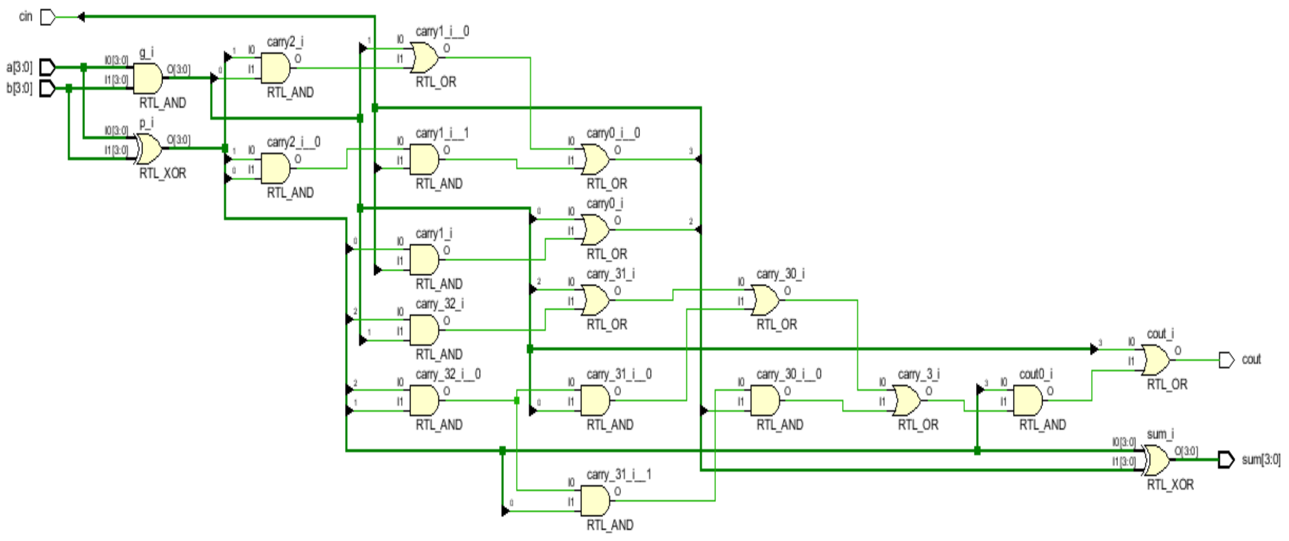


Name
Value
clk 0
reset 0
serial_in Z
serial_out 1
shift_reg[3:0] 0
[3] 0
[2] 0
[1] 0
[0] 0
2,358.005 ns
1,800.000 ns
2,000.000 ns
2,200.000 ns
2,400.000 ns
2,600.
0
Z
0
Z
0
Z

Design a Carry Look Ahead Adder using Verilog hdl

Design code:

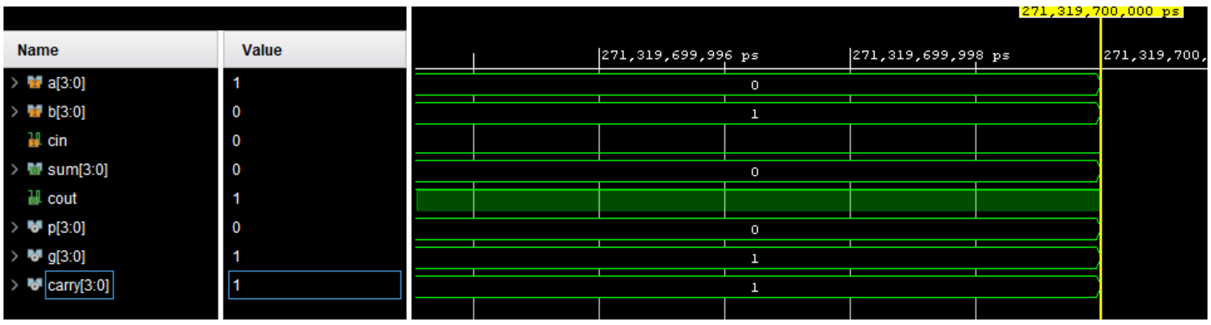
```
module claa(

   input [3:0] a,
   input [3:0] b,
   input cin,
   output [3:0] sum,
   output cout
);

   wire [3:0] p, g;
   wire [3:0] carry;


   assign p = a ^ b;
   assign g = a & b;


   assign carry[0] = cin;
   assign carry[1] = g[0] | (p[0] & cin);
   assign carry[2] = g[1] | (p[1] & g[0]) | (p[1] & p[0] & cin);
   assign carry[3] = g[2] | (p[2] & g[1]) | (p[2] & p[1] & g[0]) | (p[2] & p[1] & p[0] & cin);


   assign cout = g[3] | (p[3] & carry[3]);


   assign sum = p ^ {carry[2:0], cin};

endmodule
```



Test Bench:

```
module tb_carry_lookahead_adder_4bit;


   reg [3:0] a;
   reg [3:0] b;
   reg cin;


   wire [3:0] sum;
   wire cout;


   carry_lookahead_adder_4bit uut (
      .a(a),
      .b(b),
      .cin(cin),
      .sum(sum),
      .cout(cout)
   );


   initial begin

      a = 4'b0000;
      b = 4'b0000;
      cin = 1'b0;


      #10;

       Test Case 1: Add 0 + 0 + 0
      a = 4'b0000;
      b = 4'b0000;
      cin = 1'b0;
      #10;
      $display("TC1: a=%b, b=%b, cin=%b, sum=%b, cout=%b", a, b, cin, sum, cout);

      Test Case 2: Add 5 + 3 + 0
      a = 4'b0101;
      b = 4'b0011;
      cin = 1'b0;
      #10;
      $display("TC2: a=%b, b=%b, cin=%b, sum=%b, cout=%b", a, b, cin, sum, cout);

       Test Case 3: Add 9 + 6 + 0
      a = 4'b1001;
      b = 4'b0110;
      cin = 1'b0;
      #10;
      $display("TC3: a=%b, b=%b, cin=%b, sum=%b, cout=%b", a, b, cin, sum, cout);

      Test Case 4: Add 15 + 15 + 1
      a = 4'b1111;
```

```
    b = 4'b1111;
    cin = 1'b1;
    #10;
    $display("TC4: a=%b, b=%b, cin=%b, sum=%b, cout=%b", a, b, cin, sum, cout);

     Test Case 5: Add 8 + 7 + 1
    a = 4'b1000;
    b = 4'b0111;
    cin = 1'b1;
    #10;
    $display("TC5: a=%b, b=%b, cin=%b, sum=%b, cout=%b", a, b, cin, sum, cout);


    $finish;
  end

endmodule
```

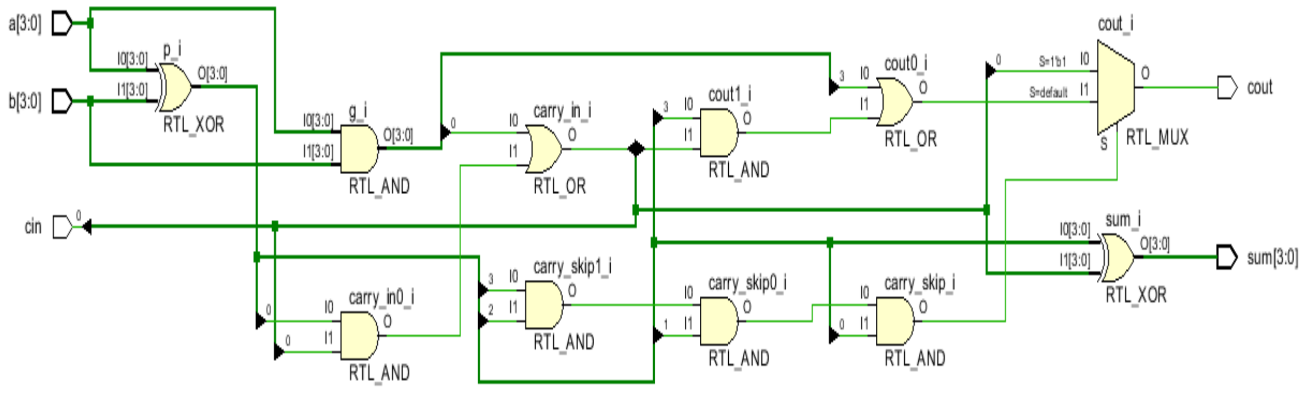


Name	Value
a[3:0]	1
b[3:0]	0
cin	0
sum[3:0]	0
cout	1
p[3:0]	0
g[3:0]	1
carry[3:0]	1

Design a Carry Skip Adder using Verilog hdl

Design Code:

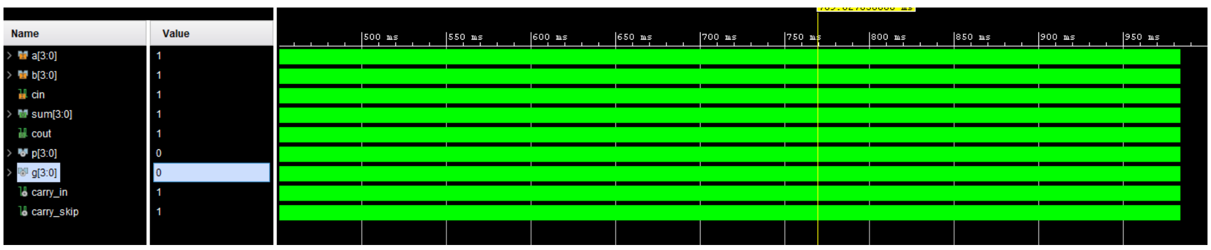
```
module csa(
   input [3:0] a,
   input [3:0] b,
   input cin,
   output [3:0] sum,
   output cout
);
   wire [3:0] p;
   wire [3:0] g;
   wire carry_in;
   wire carry_skip;
   assign p = a ^ b;
   assign g = a & b;

   assign sum = p ^ {carry_in, carry_in, carry_in, cin};
   assign carry_in = (g[0] | (p[0] & cin));
   assign carry_skip = p[3] & p[2] & p[1] & p[0];
   assign cout = (carry_skip)  cin : (g[3] | (p[3] & carry_in));

endmodule
```



Test Bench:

```
module tb_carry_skip_adder_4bit;
   reg [3:0] a;
   reg [3:0] b;
   reg cin;
   wire [3:0] sum;
   wire cout;
   carry_skip_adder_4bit uut (
      .a(a),
      .b(b),
      .cin(cin),
      .sum(sum),
      .cout(cout)
   );
   initial begin
      a = 4'b0000;
      b = 4'b0000;
      cin = 1'b0;

      #10;

       Test Case 1: Add 0 + 0 + 0
      a = 4'b0000;
      b = 4'b0000;
      cin = 1'b0;
      #10;
      $display("TC1: a=%b, b=%b, cin=%b, sum=%b, cout=%b", a, b, cin, sum, cout);

       Test Case 2: Add 5 + 3 + 0
      a = 4'b0101;
      b = 4'b0011;
```

```
        cin = 1'b0;

        #10;

        $display("TC2: a=%b, b=%b, cin=%b, sum=%b, cout=%b", a, b, cin, sum, cout);


         Test Case 3: Add 9 + 6 + 1

        a = 4'b1001;

        b = 4'b0110;

        cin = 1'b1;

        #10;

        $display("TC3: a=%b, b=%b, cin=%b, sum=%b, cout=%b", a, b, cin, sum, cout);


         Test Case 4: Add 15 + 15 + 1

        a = 4'b1111;

        b = 4'b1111;

        cin = 1'b1;

        #10;

        $display("TC4: a=%b, b=%b, cin=%b, sum=%b, cout=%b", a, b, cin, sum, cout);


        Test Case 5: Add 8 + 7 + 0

        a = 4'b1000;

        b = 4'b0111;

        cin = 1'b0;

        #10;

        $display("TC5: a=%b, b=%b, cin=%b, sum=%b, cout=%b", a, b, cin, sum, cout);



        $finish;

    end


endmodule
```



Name	Value
a[3:0]	1
b[3:0]	1
cin	1
sum[3:0]	1
cout	1
p[3:0]	0
g[3:0]	0
carry_in	1
carry_skip	1