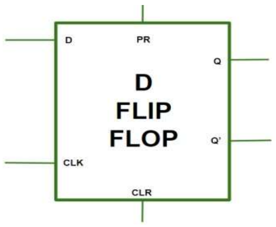
Experiment : Flipflops

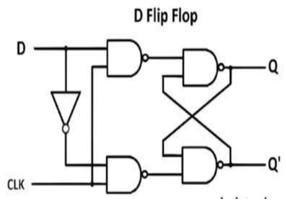
D-Flipflop:

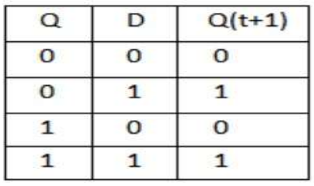
Aim: For D- Flipflop.

Write Verilog description

Test-bench



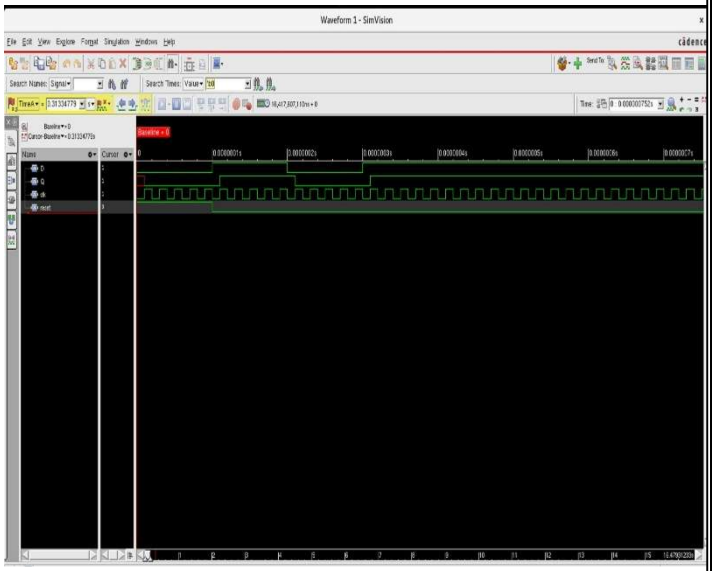




Q	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

Verilog Code

```
module d_ff (
   input clk,
   input d,
   output reg q
);
   always @(posedge clk) begin
      q <= d;
   end
endmodule
```



Testbench

```
module tb_d_ff;
   reg clk, d;
   wire q;
 d_ff uut (.clk(clk), .d(d), .q(q));
initial begin
 clk = 0; d = 0;
      #10 d = 1;
```

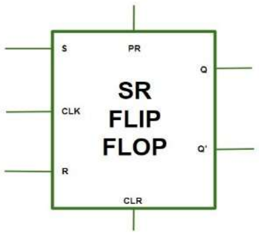
```
        #10 d = 0;

        #10 d = 1;

        #50 $finish;

    End

always #5 clk = ~clk;  // Clock generation

endmodule
```

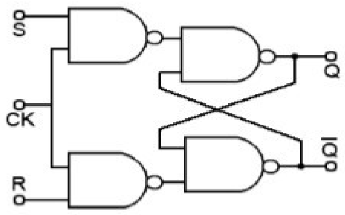
SR- Flipflop:

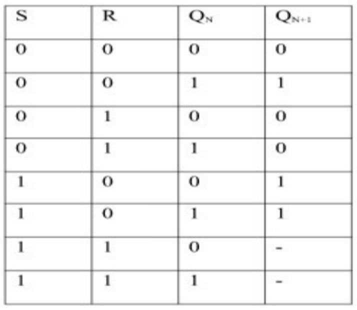
Aim : For SR- Flipflop.

Write Verilog description

Test-bench

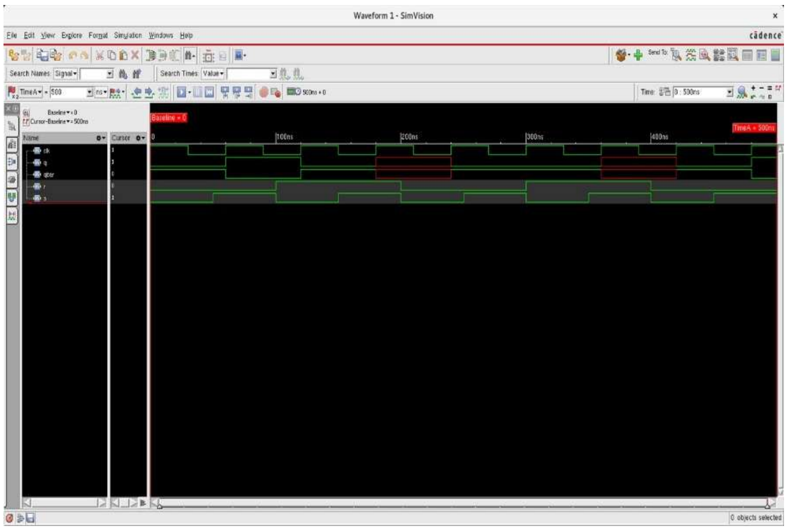






S	R	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	-

```
module sr_ff (

    input clk,

    input s,

    input r,

    output reg q

);

    always @(posedge clk) begin

        case ({s, r})

            2'b00: q <= q;

            2'b01: q <= 0;

            2'b10: q <= 1;

            2'b11: q <= 1'bx;

        endcase

    end      endmodule
```



Testbench

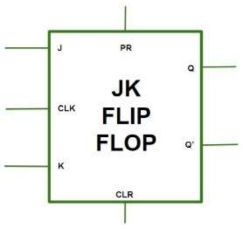
```
module tb_sr_ff;
    reg clk, s, r;
    wire q;
  sr_ff uut (.clk(clk), .s(s), .r(r), .q(q));
      always #5 clk = ~clk;
           initial begin
        clk = 0; s = 0; r = 0;
        #10 s = 1; r = 0;
        #10 s = 0; r = 1;
        #10 s = 0; r = 0;
        #10 s = 1; r = 1;
        #10 s = 0; r = 0;
        #10 $stop;
    end
endmodule
```

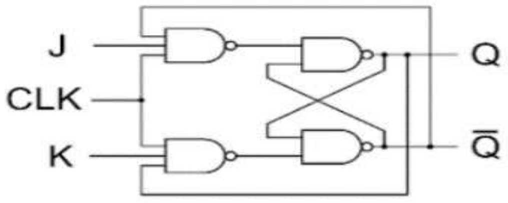
JK Flipflop

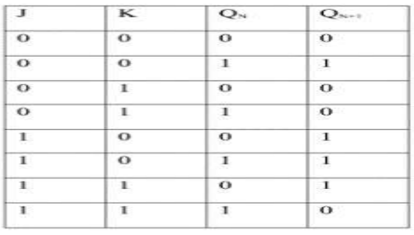
Aim: For JK- Flipflop

Write Verilog description

Test-bench

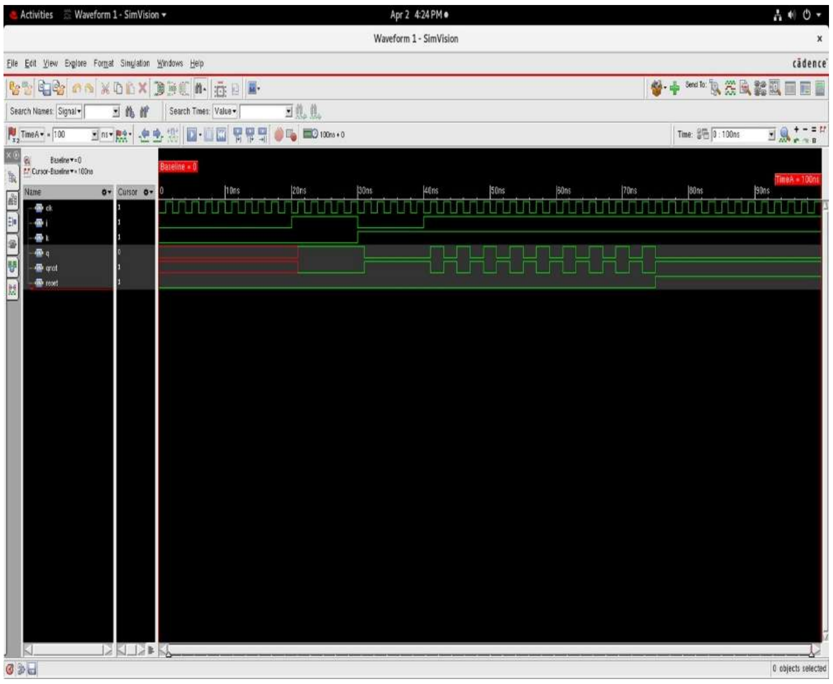






J	K	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

```
module jk_ff (
    input clk,
    input j,
    input k,
    output reg q
);
    always @(posedge clk) begin
        case ({j, k})
            2'b00: q <= q;
            2'b01: q <= 0;
            2'b10: q <= 1;
            2'b11: q <= ~q;
        endcase
    end
endmodule
```



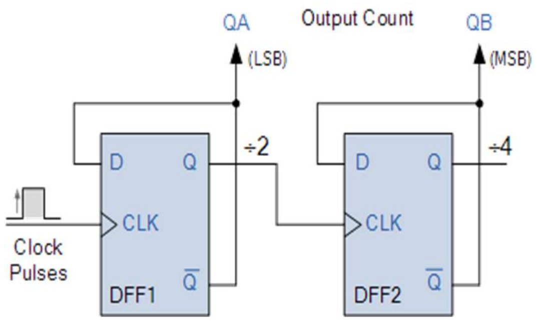
Testbench

```
module tb_jk_ff;
    reg clk, j, k;
    wire q;
jk_ff uut (.clk(clk), .j(j), .k(k), .q(q));
always #5 clk = ~clk;  // Clock toggles every 5ns
  initial begin
        clk = 0; j = 0; k = 0;
        #10 j = 1; k = 0;
        #10 j = 0; k = 1;
        #10 j = 1; k = 1;
        #10 j = 0; k = 0;  //
#10 $finish;
    end
endmodule
```

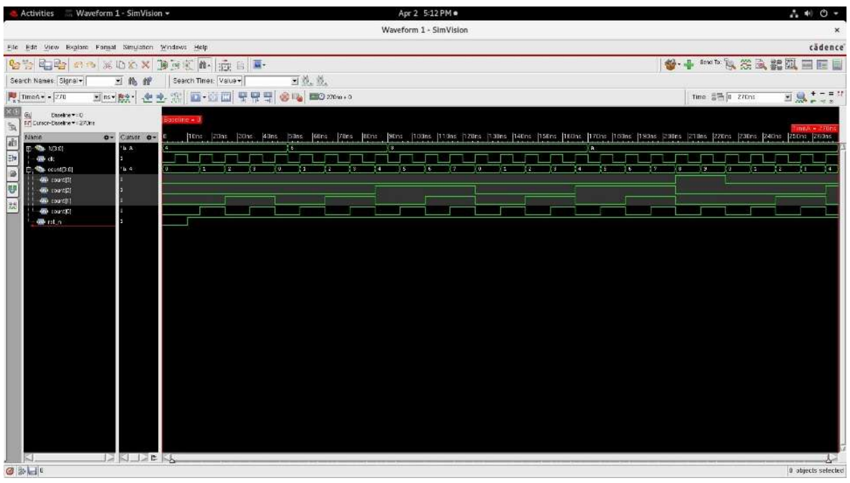
Experiment : MOD N Counter

Aim: To design a four bit Synchronous MOD-N counter with Asynchronous reset

Write Verilog Code



```
module mod_n_counter #(parameter N = 10, WIDTH = 4)(
    input clk,
    input rst,
    output reg [WIDTH-1:0] count
);
    always @(posedge clk or posedge rst) begin
        if (rst)
            count <= 0;
        else if (count == N-1)
            count <= 0;
        else
            count <= count + 1;
    end
endmodule
```



Testbench

```
module tb_mod_n_counter;
    parameter N = 10;
    reg clk = 0, rst;
    wire [3:0] count;
 mod_n_counter #(N, 4) uut (.clk(clk), .rst(rst), .count(count));
```

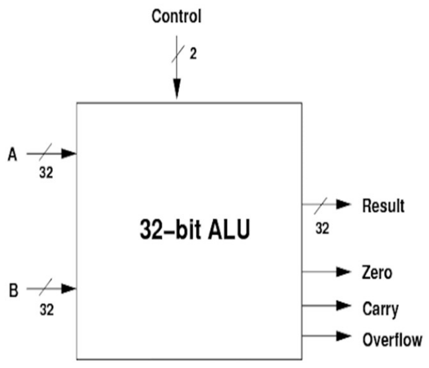
```
      always #5 clk = ~clk;
       initial begin
        rst = 1; #10;
        rst = 0;
         #100;
        $finish;
    end
endmodule
```

Experiment : 32 bit ALU

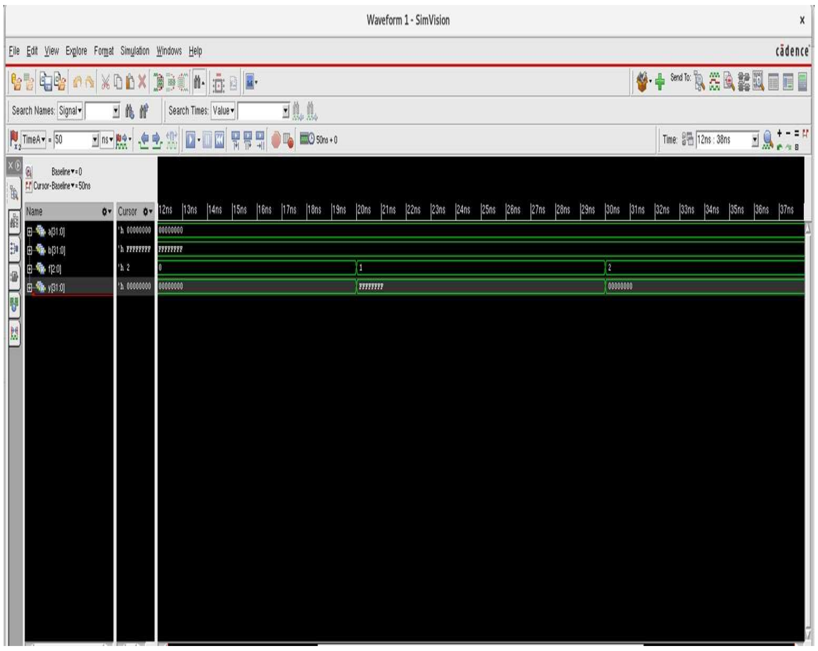
Aim: 32-Bit ALU Supporting 4-Logical and 4-Arithmetic operations, using case and if statement for

ALU Behavioral Modeling

Write Verilog description



```
module alu_32bit (
    input [31:0] a,
    input [31:0] b,
    input [2:0] sel,
    output reg [31:0] result,
    output zero
);
    always @(*) begin
        case (sel)
            3'b000: result = a + b;
            3'b001: result = a - b;
            3'b010: result = a & b;
            3'b011: result = a | b;
            3'b100: result = a ^ b;
            3'b101: result = ~a;
            3'b110: result = a << 1;
            3'b111: result = a >> 1;
            default: result = 32'h00000000;
        endcase
    end
assign zero = (result == 0) ? 1 : 0;
endmodule
```

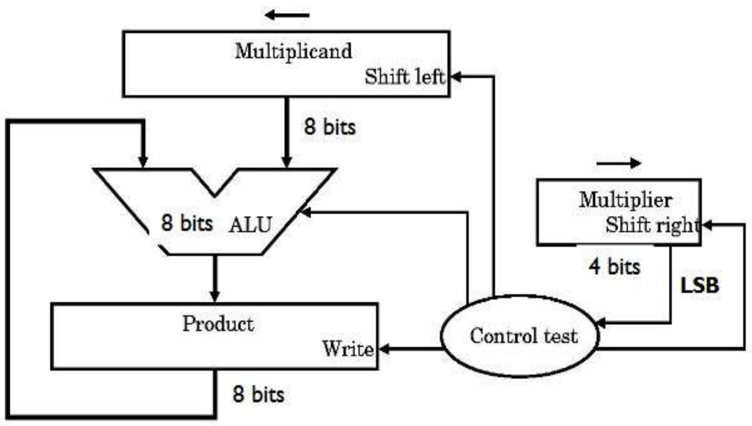

Waveform 1 - SimVision
File Edit View Explore Format Simulation Windows Help
cādence
Search Names: Signal
Search Times: Value
TimeA = 50 ns
50ns + 0
Time: 12ns : 38ns
Baseline = 0
Cursor-Baseline = 50ns
Name
Cursor
12ns 13ns 14ns 15ns 16ns 17ns 18ns 19ns 20ns 21ns 22ns 23ns 24ns 25ns 26ns 27ns 28ns 29ns 30ns 31ns 32ns 33ns 34ns 35ns 36ns 37ns
a[31:0] 'h 00000000 00000000
b[31:0] 'h FFFFFFFF FFFFFFFF
f[2:0] 'h 2 0 1 2
y[31:0] 'h 00000000 00000000 FFFFFFFF 00000000

Testbench

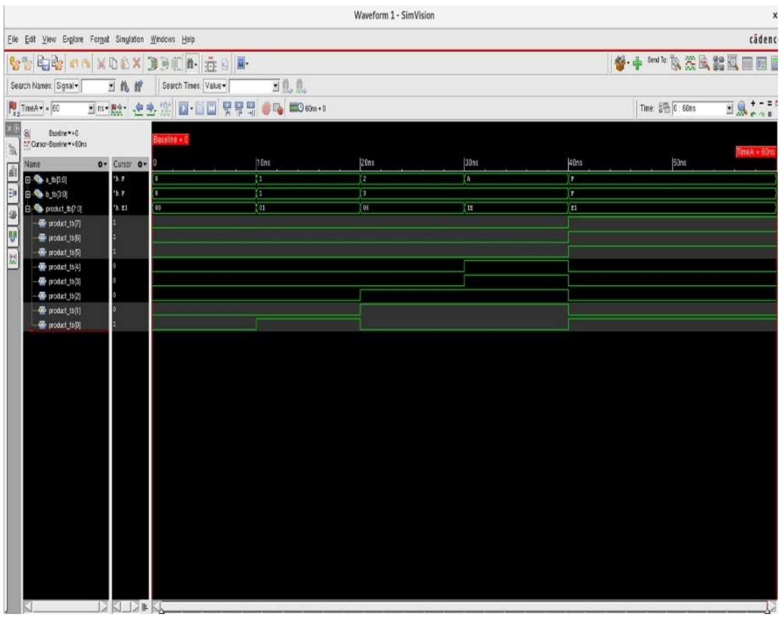
```
module tb_alu_32bit;
    reg [31:0] a, b;
    reg [2:0] sel;
    wire [31:0] result;
    wire zero;
 alu_32bit uut (.a(a), .b(b), .sel(sel), .result(result), .zero(zero));
  initial begin
        a = 32'd15; b = 32'd10;
        sel = 3'b000; #10;
        sel = 3'b001; #10;
        sel = 3'b010; #10;
        sel = 3'b011; #10;

        sel = 3'b100; #10;

        sel = 3'b101; #10;

        sel = 3'b110; #10;

        sel = 3'b111; #10;


        #10 $finish;

    end

endmodule
```

Experiment : 4bit Shift and Add Multiplier

Aim: To write a verilog code for 4bit shift and add multiplier and verify the functionality using Test bench.



```
module shift_and_mult_4bit (
    input [3:0] a,
    input [3:0] b,
    input [1:0] sel ,
    output reg [3:0] out
);
    always @(*) begin
        case (sel)
            2'b00: out = a;
            2'b01: out = a << 1;
            2'b10: out = a >> 1;
            2'b11: out = a & b;
            default: out = 4'b0000;
        endcase
    end
endmodule
```

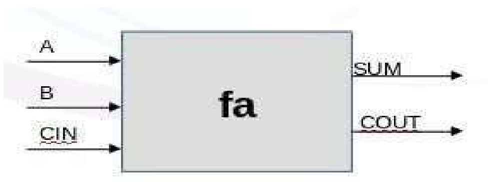


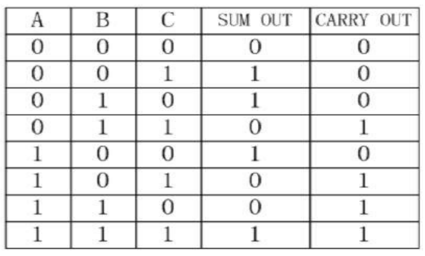
Testbench

```
module tb_shift_and_mult_4bit;
    reg [3:0] a, b;
    reg [1:0] sel;
    wire [3:0] out;

    shift_and_mult_4bit uut (.a(a), .b(b), .sel(sel), .out(out));

    initial begin
        a = 4'b1010; b = 4'b1100;
       sel = 2'b00; #10;
        sel = 2'b01; #10;
        sel = 2'b10; #10;

        sel = 2'b11; #10 ;


        #10 $finish;

    end

endmodule
```

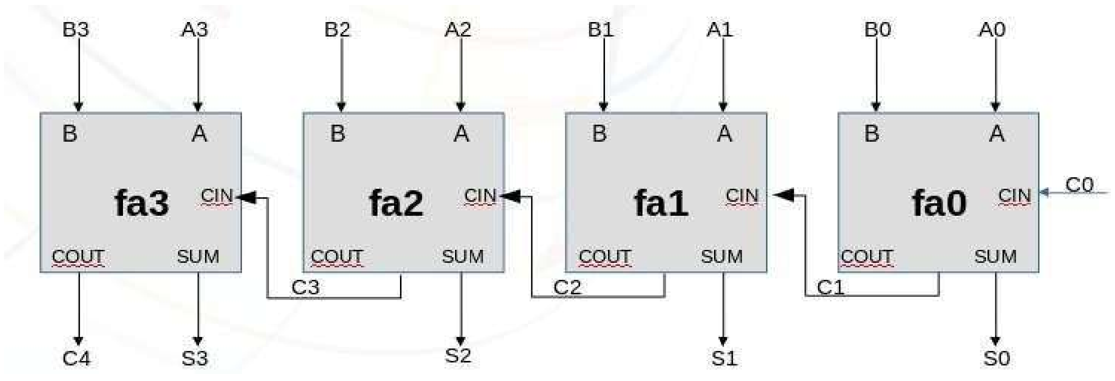
Experiment : 4 Bit adder

Aim: To write a verilog code for 4bit adder and verify the functionality using Test bench





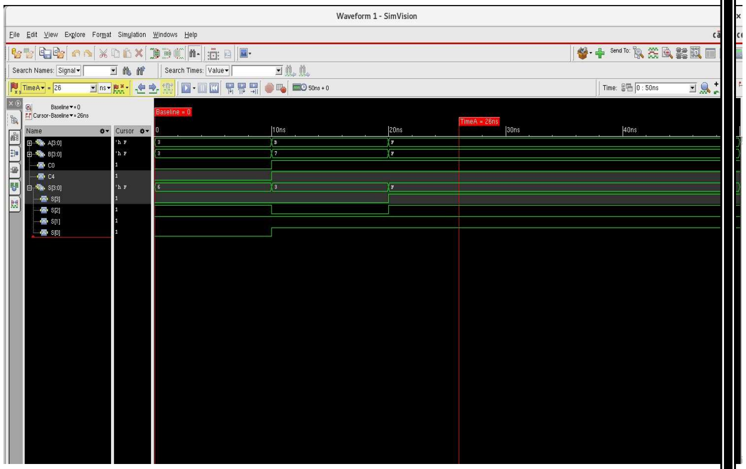
A	B	C	SUM OUT	CARRY OUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



```
module adder_4bit (

  input [3:0] a,

  input [3:0] b,

  input cin,
  output [3:0] sum,

  output cout

);

  wire c1, c2, c3;


  full_adder FA0 (.a(a[0]), .b(b[0]), .cin(cin), .sum(sum[0]), .cout(c1));

  full_adder FA1 (.a(a[1]), .b(b[1]), .cin(c1),  .sum(sum[1]), .cout(c2));

  full_adder FA2 (.a(a[2]), .b(b[2]), .cin(c2),  .sum(sum[2]), .cout(c3));

  full_adder FA3 (.a(a[3]), .b(b[3]), .cin(c3),  .sum(sum[3]), .cout(cout));

endmodule


module full_adder (

  input a, b, cin,

  output sum, cout

);

  assign sum  = a ^ b ^ cin;

  assign cout = (a & b) | (b & cin) | (a & cin);

endmodule
```

Testbench

```
module tb_adder_4bit;

    reg [3:0] a, b;

    reg cin;

    wire [3:0] sum;

    wire cout;


    adder_4bit uut (.a(a), .b(b), .cin(cin), .sum(sum),
.cout(cout));


    initial begin

        a = 4'b0011; b = 4'b0101; cin = 0; #10;
```



```
        a = 4'b1111; b = 4'b0001; cin = 0; #10;  //
        a = 4'b1010; b = 4'b0101; cin = 1; #10;  //
        #10 $finish;
    end
endmodule
```

OR

```
module adder_4bit (
    input [3:0] a,
    input [3:0] b,
    input cin,
    output [3:0] sum,
    output cout
);
    assign {cout, sum} = a + b + cin;
endmodule
```

Testbench

```
module tb_adder_4bit;
    reg [3:0] a, b;
    reg cin;
    wire [3:0] sum;
    wire cout;

    adder_4bit uut (.a(a), .b(b), .cin(cin), .sum(sum), .cout(cout));

    initial begin
```

```
        a = 4'd3; b = 4'd5; cin = 0; #10;

        a = 4'd15; b = 4'd1; cin = 0; #10;

        a = 4'd10; b = 4'd5; cin = 1; #10;

        #10 $finish;

    end

endmodule
```