4 X 1 - Multiplexer

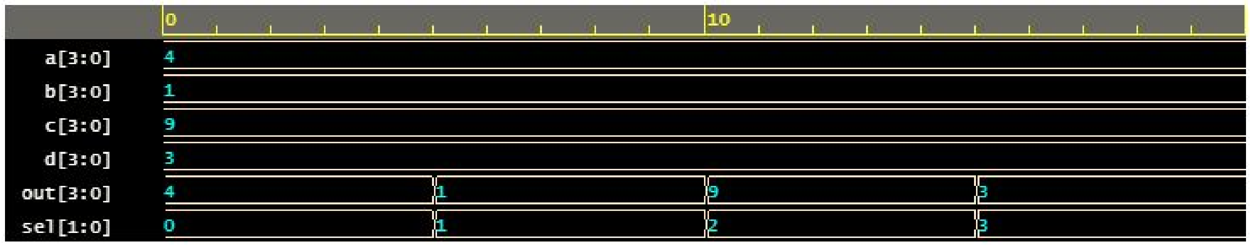
Aim: To design 4x1 multiplexer using Xilinx software

Apparatus: Xilinx software, Computer

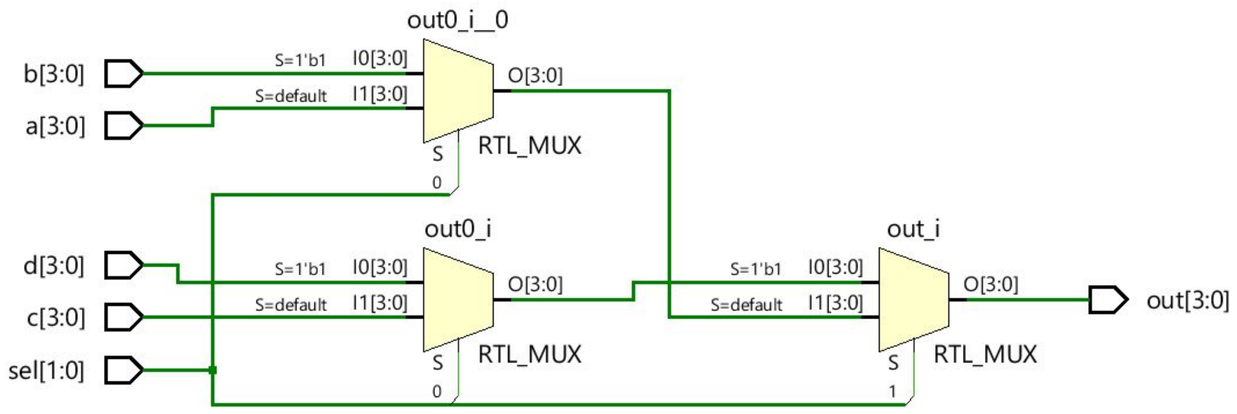
Code:

```
module mux_4to1_assign ( input [3:0] a,                 // 4-bit input called a
                         input [3:0] b,                 // 4-bit input called b
                         input [3:0] c,                 // 4-bit input called c
                         input [3:0] d,                 // 4-bit input called d
                         input [1:0] sel,               // input sel used to select between a,b,c,d
                         output [3:0] out);             // 4-bit output based on input sel

  // When sel[1] is 0, (sel[0]? b:a) is selected and when sel[1] is 1, (sel[0] ? d:c) is taken
  // When sel[0] is 0, a is sent to output, else b and when sel[0] is 0, c is sent to output,
else d
  assign out = sel[1] ? (sel[0] ? d : c) : (sel[0] ? b : a);

endmodule
```

RTL Schematic:



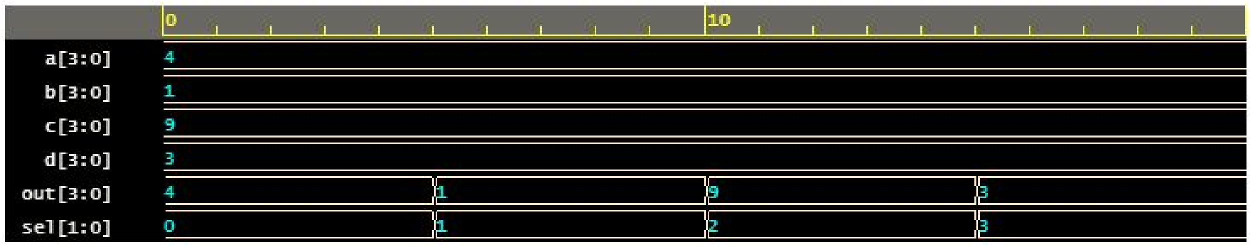
Hardware Schematic:



Testbench:

```
module tb_4to1_mux;

  // Declare internal reg variables to drive design inputs
  // Declare wire signals to collect design output
  // Declare other internal variables used in testbench
  reg [3:0] a;
  reg [3:0] b;
  reg [3:0] c;
  reg [3:0] d;
  wire [3:0] out;
  reg [1:0] sel;
  integer i;

  // Instantiate one of the designs, in this case, we have used the design with case
statement
  // Connect testbench variables declared above with those in the design
  mux_4to1_case  mux0 (   .a (a),
                          .b (b),
                          .c (c),
                          .d (d),
                          .sel (sel),
                          .out (out));

  // This initial block is the stimulus
  initial begin
    // Launch a monitor in background to display values to log whenever a/b/c/d/sel/out
changes
    $monitor ("[%0t] sel=0x%0h a=0x%0h b=0x%0h c=0x%0h d=0x%0h out=0x%0h",
$time, sel, a, b, c, d, out);

     // 1. At time 0, drive random values to a/b/c/d and keep sel = 0
    sel <= 0;
    a <= $random;
    b <= $random;
    c <= $random;
    d <= $random;

    // 2. Change the value of sel after every 5ns
    for (i = 1; i < 4; i=i+1) begin
      #5 sel <= i;
    end

    // 3. After Step2 is over, wait for 5ns and finish simulation
    #5 $finish;
  end
endmodule
```

Simulation:



Result: Thus the simulation and design of 4x1 multiplexer is studied using Xilinx software.

Verilog Full Adder

Aim: To design verilog full adder using Xilinx software

Apparatus: Xilinx software, Computer

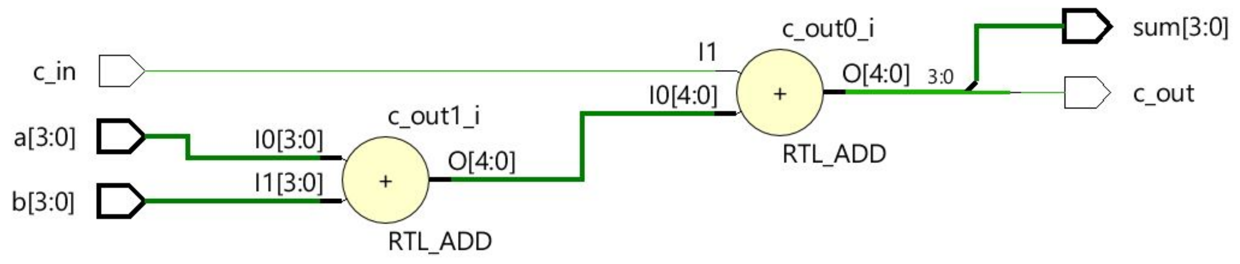
Truth Table

A	B	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Code:

```
module fulladd (  input [3:0] a,
                  input [3:0] b,
                  input c_in,
                  output c_out,
                  output [3:0] sum);

   assign {c_out, sum} = a + b + c_in;
endmodule
```

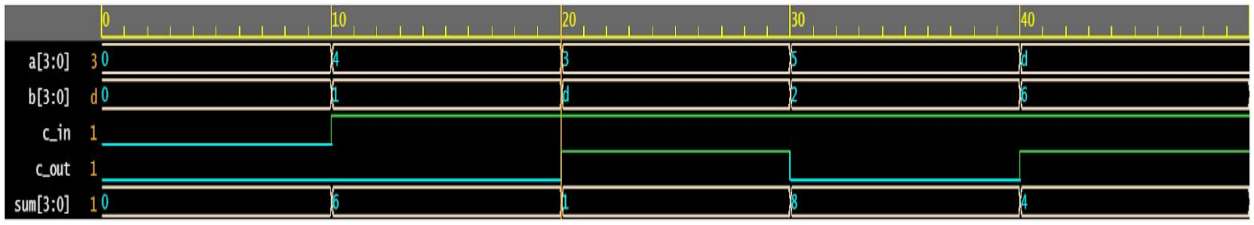
Hardware Schematic:



Testbench:

```
module tb_fulladd;
 // 1. Declare testbench variables
  reg [3:0] a;
  reg [3:0] b;
  reg c_in;
  wire [3:0] sum;
  integer i;

 // 2. Instantiate the design and connect to testbench variables
  fulladd  fa0 ( .a (a),
                 .b (b),
                 .c_in (c_in),
                 .c_out (c_out),
                 .sum (sum));

 // 3. Provide stimulus to test the design
  initial begin
    a <= 0;
    b <= 0;
    c_in <= 0;

    $monitor ("a=0x%0h b=0x%0h c_in=0x%0h c_out=0x%0h sum=0x%0h", a, b, c_in,
c_out, sum);

   // Use a for loop to apply random values to the input
    for (i = 0; i < 5; i = i+1) begin
      #10 a <= $random;
          b <= $random;
           c_in <= $random;
    end
  end
endmodule
```

RTL Schematic:



Result: Thus study and design of Full adder counter is implemented using Xilinx software.

4 bit UP/DOWN Counter

Aim: To study design 4 bit Up/Down Counter using Xilinx software

Apparatus: Xilinx software, Computer

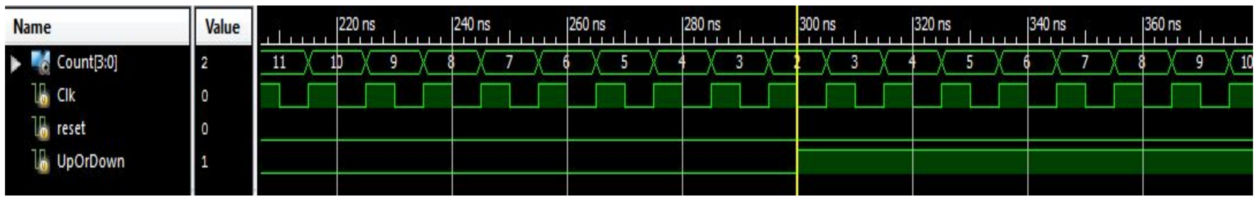
Code:

```
//Verilog module for UpDown counter
//When Up mode is selected, counter counts from 0 to 15 and then again from 0
to 15.
//When Down mode is selected, counter counts from 15 to 0 and then again from
15 to 0.
//Changing mode doesn't reset the Count value to zero.
//You have apply high value to reset, to reset the Counter output.
module upordown_counter(
    Clk,
    reset,
    UpOrDown,  //high for UP counter and low for Down counter
    Count
    );


    //input ports and their sizes
    input Clk,reset,UpOrDown;
    //output ports and their size
    output [3 : 0] Count;
    //Internal variables
    reg [3 : 0] Count = 0;

     always @(posedge(Clk) or posedge(reset))
     begin
        if(reset == 1)
            Count <= 0;
        else
            if(UpOrDown == 1)   //Up mode selected
                if(Count == 15)
                    Count <= 0;
                else
                    Count <= Count + 1; //Incremend Counter
            else  //Down mode selected
                if(Count == 0)
                    Count <= 15;
                else
                    Count <= Count - 1; //Decrement counter
     end

endmodule
```

Testbench for counter:

```
module tb_counter;

    // Inputs
    reg Clk;
    reg reset;
    reg UpOrDown;

    // Outputs
    wire [3:0] Count;

    // Instantiate the Unit Under Test (UUT)
    updown_counter uut (
        .Clk(Clk),
        .reset(reset),
        .UpOrDown(UpOrDown),
        .Count(Count)
    );

//Generate clock with 10 ns clk period.
    initial Clk = 0;
    always #5 Clk = ~Clk;

    initial begin
        // Apply Inputs
        reset = 0;
        UpOrDown = 0;
        #300;
        UpOrDown = 1;
      #300;
        reset = 1;
        UpOrDown = 0;
        #100;
        reset = 0;
    end

Endmodule
```

RTL Schematic:



Result: Thus study and design of Up/Down counter is implemented using Xilinx software.